Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Synthesis and Stochastic Assessment of Cost-Optimal Schedules

Angelika Mader[1], Henrik Bohnenkamp[2], Yaroslav S. Usenko[3], David N. Jansen[4], Johann Hurink[1], Holger Hermanns[5,1]

[1] University of Twente
Faculty EWI, 7500 AE Enschede, The Netherlands
e-mail: {a.h.mader|j.l.hurink}@utwente.nl
[2] RWTH Aachen University
Modeling and Verification of Software, 52056 Aachen, Germany
e-mail: henrik@cs.rwth-aachen.de
[3] Technical University of Eindhoven
Laboratory for Quality Software (LaQuSo), 5600 MB Eindhoven, The Netherlands
e-mail: Y.S.Usenko@tue.nl
[4] Radboud University Nijmegen
Computing Science Institute
e-mail: d.jansen@cs.ru.nl
[5] Saarland University
Dependable Systems and Software Group, 66041 Saarbrücken, Germany
e-mail: hermanns@cs.uni-sb.de

The date of receipt and acceptance will be inserted by the editor

Abstract. We treat the problem of generating cost-optimal schedules for orders with individual due dates and cost functions based on earliness/tardiness. Orders can run in parallel in a resource-constrained manufacturing environment, where resources are subject to stochastic breakdowns. The goal is to generate schedules while minimizing the expected costs. First, we estimate the distribution of each order type by simulation (assuming a reasonable machine/load model) and derive from the cost-function an optimal offset from the due date of each individual order. Second, these optimal offsets are then used to guide the generation of schedules which are responsible to resolve resource conflicts. Third, we evaluate the generated schedules by simulation. The approach is demonstrated by means of a non-trivial case-study from lacquer production. Optimal offsets are derived with the MODEST/MÖBIUS tool, schedules are generated using UPPAAL CORA. The experimental results show that our approach achieves good results in all considered scenarios, and better results than an approach based on adding slack to processing times.

Key words: Model Checking, Scheduling, Heuristic Methods, Discrete Event Simulation

1 Introduction

Scheduling problems arise in a wide range of different application areas. They have motivated cross-disciplinary research for a long time, notably between operations research and computer science. Many scheduling problems are nowadays well-understood, and depending on the type, solutions can be obtained sometimes with analytic-exact or at least with heuristic methods. Their solution is usually based on techniques from classical scheduling theory, which is a well-established branch of operations research [16,20]. In particular, industrial scheduling problems arising in manufacturing are highly complex, such that only heuristic solutions can be used in practice. Recently, the application of search techniques from model checking and mixed integer linear programming have been proposed as a complementary approach to schedule synthesis [21]. The potential advantage of this approach lies in its simplicity and generality. Where classical scheduling solutions often rely on restricting assumptions that may not be fulfilled in practice, state space exploration techniques are generally applicable, and very robust under variation in the problem parameters. In the model checking approach scheduling feasibility is expressed as temporal or real-time property ("The production can be finished until time T."), and a model checker is used to, first, check whether this property holds for a reachable state in a model of the overall system behavior, and second, to return a path through the state-space of the model which represents the schedule which accomplishes the task. More recently, this approach has been further developed to use models with superimposed costs and reward structure. Schedules can then be improved by minimizing a given cost-function [11].

We address a scheduling problem characterized by earliness/tardiness constraints and machine breakdowns. *Earliness/tardiness* means that finishing an order *before* the due date causes storage costs, and finishing it *after* the due date

causes delay penalties[1]. Machine breakdowns are by nature stochastic events that make deterministic scheduling, without taking the stochastic effects into account, ineffective. We consider a concrete case-study originating from a scheduling problem in lacquer production. Lacquers are produced according to certain *recipes*. A recipe comprises different, non-preemptive tasks which are ordered according to a precedence relation with timing constraints. Each task has a fixed duration and requires one or more renewable resources of different types. Resources are prone to failure, but are repairable. The failure behavior is described by the so-called performance factors p, which are essentially defined as the ratio $p = \mathsf{uptime}/(\mathsf{uptime} + \mathsf{downtime})$ of a resource. Only p is assumed to be known, but nothing about uptime and downtime. Execution of a task, interrupted by a breakdown, is resumed after a repair. Therefore, the effective task durations are of a stochastic nature, since we assume the breakdown/repair behavior being of a stochastic nature. Lacquers are *ordered* for certain due dates, and several orders can be in productions at the same time. The resource requirements within a single order are conflict free, but the tasks of different orders are competing for the resources available.

The case taken here as example was originally introduced within the AMETIST project [5]. The setting we used is the same. However, in the AMETIST project it was assumed that the approach with extended processing times would give the best results for schedules. Doubting this assumption lead to the initial idea for this contribution.

In the literature, uncertainty in job scheduling occurs in two different ways: either the data of the orders/jobs is uncertain (e.g. processing times or due dates) or the machines/resources may fail (for an overview see e.g. [25]). The considered problem clearly falls into the second class. For this class the existing solution approaches roughly can be divided in two groups. The first group explicitly deals with the probability distribution of the breakdowns. However, mainly analytic results for relatively simple problems (compared to our setting) are achieved. For example, [17] treat the asymmetric earliness/tardiness problem for a single machine only.

The second group consist of approaches that deal with the breakdowns either by adding idle time into a predictive schedule (see e.g. [30,29]), or by enlarging the processing times dependent on the probability of a breakdown for the corresponding resource (see e.g. [22]). In this last category falls the following approach, which, due to its simplicity, is used quite frequently in practice. The idea is to *extend* the task duration, *i. e.,* introduce slack to allow for possible breakdowns. The task duration d_t of task t is extended by factor $1/p$, where p is the performance factor of the resource on which t is processed. Schedules are then generated taking these *extended processing times* (EPT) as a basis, rather than the original task durations. We call this approach the *EPT approach*.

[1] In the literature orders are often called *jobs*. We use the terms interchangeably.

In this paper, however, we suggest a different solution to schedule generation in the described scenario. The basic idea is to combine *stochastic simulation* and *model checking* in an effective manner. Technically, we first use stochastic simulation to determine an *optimal starting time* (OST) for each job. The result is a heuristic estimate minimizing the expected costs, and is based on a stochastic machine breakdown model and a load model which approximates the precise resource conflicts. That simulation is carried out using the tools MODEST/MÖBIUS [14,13,31]. Second, we synthesize schedules based on these optimal starting times using the timed automaton model checker UPPAAL CORA [36]. In this step the actual resource conflicts are solved while trying to approximate the optimal starting times as well as possible. We name this approach to schedule synthesis the *OST approach*.

The results of the OST approach are evaluated using stochastic simulation. We assume several sets of distributions to describe *time-to-failure* and *time-to-repair* of the different resources. In practice, the relevant information of a schedule is the starting times of the orders. In case a stochastic event invalidates the precomputed schedules, only the ordering of tasks on machines is followed, not the precise timing. We follow this strategy in the simulative evaluation, and address two aspects. First, we compare our approach against the EPT approach mentioned above. Second, we investigate how stable our approach is with respect to erroneous assumptions about the chosen distributions. Since in practice precise distributions are unlikely to be known, we will always have to deal with an approximation which is to a certain extent erroneous.

Our setting has several features which, to our knowledge, have not been addressed in this combination in the relevant literature. First, we assume multiple recipes of different types, and assume concurrent orders with different cost functions. In the literature, orders often are executed in sequence, which allows the orders to be considered in isolation(*e. g.,* [24]). Furthermore, we treat orders and resources separately by using a machine/load model. As a consequence, the breakdown behavior of machines is only part of the machine model, and the effective task durations resulting from breakdowns and repairs are stochastically dependent. This gives a more accurate description of reality. Usually, in the literature task durations are modeled by independent random variables (*e. g.,* [34]). Regarding the objective function, the combination of earliness and tardiness leads to a non-regular cost-function implying that restricting to active schedules (i.e. not allowing to wait for otherwise unused resources, see e.g. [33]) no longer guarantees optimality. This fact makes it much harder or even pointless to apply priority based dispatching approaches. Finally, to our knowledge, simulation for evaluation of scheduling techniques is not a standard approach.

A model which is close to the one presented in this paper, but which has some of the restrictions mentioned above, is used by Stork [34], who considers single jobs with independent stochastic task durations and resource constraints. The main objective there is to define scheduling *policies* with the objective to minimize the cost function. This is thus an approach to *reactive* scheduling, whereas our approach is proac-

tive [24]. Lambrechts et al. [27] pick up the model of Stork, but are indeed interested in proactive schedules. The objective is in spirit thus the closest to that of this paper, whereas the method is completely different: in [27], a classical optimization method (tabu search) is employed.

In a previous paper [12], we have investigated schedules in the lacquer production framework with respect to feasibility, *i.e.*, whether orders are finished before their due dates.

Structure of the paper. In Section 2 we describe the case study that we use to demonstrate our approach. In Section 3 we describe UPPAAL CORA and MODEST/MÖBIUS. In Section 4 we describe our approach to schedule synthesis. In Section 5 we describe the experiments conducted to evaluate our approach. We conclude with Section 6.

2 Lacquer Production Case

The purpose of this section is to present the necessary details of the lacquer-production case study, which leads to a non-trivial scheduling problem with several complex aspects. Research on this case-study has already led to several publications [12,32,7,8]. The case study was proposed by the german company *Axxom*, that specializes on software tools for planning and logistics.

As a test-bed for our investigation we will make use of a scenario based on a lacquer production plant. This scenario was originally used as a case-study in the European IST project Advanced Methods for Timed Systems (AMETIST) [5].

There are three recipes for different types of lacquer: *metallic*, *bronze* and *uni* lacquers. Each recipe describes the processing steps, their durations, their interdependency, precedence relations with timing constraints, and finally, the resources to use. A recipe is composed of *tasks*. Each task describes which resource it requires, and for how long that resource is needed. The latter time value is called the *processing time* of the task, or task duration. Figure 1 contains a graphical representation of the three considered recipes. The rectangular objects represent the tasks. The patterns indicate the resource type needed to execute the task, and the numbers the processing time in minutes. The tasks related to mixing vessels are implicit, since they do not have pre-specified processing times but last as long as the parallel tasks are performed. Vertical lines denote dependencies between the tasks (top to bottom), and horizontal lines represent synchronization. In the case no numbers are given next to the vertical lines, the dependencies express ordinary precedence constraints between the tasks, indicating that the later task may only start its processing after the former task has finished. If an interval $[a, b]$ is given next to the vertical line, this indicates that the later task has to start later than the preceding task by an amount in range between a and b.

An *order* of a certain type of lacquer is specified by the type of lacquer to be produced, the *due date* until which it has to be finished, and a cost function. An order that is finished before the due date will cause storage costs, finishing

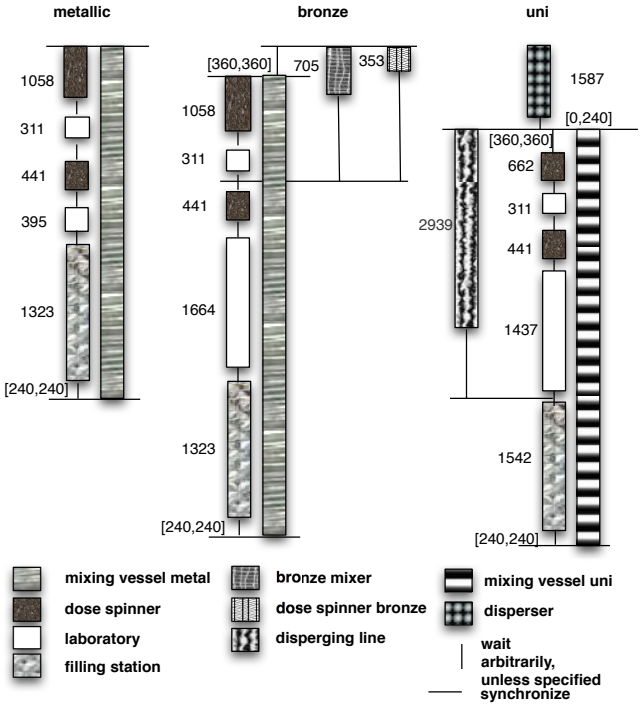

Fig. 1. Lacquer recipes indicating the resources needed, durations of processing steps and precedence relations with timing constraints

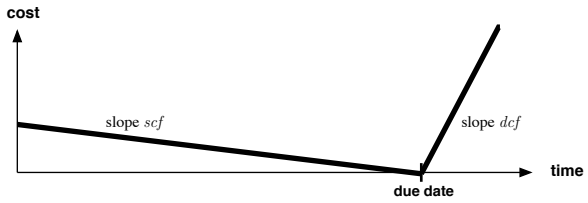

Fig. 2. Example cost function for product finish time

after the due date causes delay costs. We assume that costs are incurred linearly with time, *i.e.*, the storage cost and delay cost can be described by a piecewise linear function with two parameters scf and dcf: the *storage cost factor* and the *delay cost factor*. As a result, the costs $c(d, t)$ of finishing a job with due date d at time t is given by

$$c(d,t) = \begin{cases} scf(d-t) \text{ if } t \leq d \\ dcf(t-d) \text{ if } t > d. \end{cases}$$

Figure 2 contains a schematic representation of the cost function with respect to the due date. An order finished precisely at the due date will cause zero cost.

The concrete case from practice, which is considered in this paper, has a fixed set of 29 orders. In Table 1, the parameters for the different orders are summarized. Note that for most orders, $scf < dcf$, and the relation dcf/scf is in the order of 30 to 60. This observation will become important in Section 4. However, for some few orders, $scf > dcf$ holds.

As can be seen in Figure 1, there are nine different types of resources. These resources are subject to failure. The failure behavior of resource type r is described by a *performance*

order	type	due	*scf*	*dcf*
1	metal	27444	8	429
2	metal	33348	8	390
3	metal	33636	11	574
4	metal	36372	8	425
5	metal	37092	10	496
6	metal	40500	11	584
7	metal	41699	15	779
8	metal	47172	10	531
9	metal	56964	10	492
10	metal	57252	14	708
11	metal	57539	14	691
12	metal	62148	7	368
13	metal	67764	6	283
14	metal	67980	7	354
15	metal	71868	4	213
16	bronze	43500	17	9
17	bronze	50580	15	744
18	bronze	60660	10	531
19	bronze	61859	13	673
20	bronze	65100	8	4
21	uni	30459	3	177
22	uni	41628	3	2
23	uni	43764	7	354
24	uni	44004	6	283
25	uni	50580	10	531
26	uni	60588	7	4
27	uni	61788	10	5
28	uni	77124	8	390
29	uni	85764	8	425

Table 1. The parameters for the 29 considered orders

factor p_r. Intuitively, the performance factor is the ratio of $\mathsf{uptime}/(\mathsf{uptime} + \mathsf{downtime})$, where uptime and downtime are the up- and downtime of the resource. The values uptime and downtime must actually be considered as random variables, since the failure of a resource is a stochastic phenomenon. However, no information is available about the mean durations of up- and downtime, much less the distributions, which makes it necessary to make later certain assumptions on the distributions.

3 The tool chain

To derive and assess schedules, two tools are used. Schedules are synthesized with the model-checking tool UPPAAL CORA [36], an extension of UPPAAL [35,9,6]. The production plant and the different orders are specified in terms of *priced timed-automata*. UPPAAL generates the state-space underlying this specification and finds a path through the state space which implicitly describes a schedule leading to minimal cost. Since the generated state-spaces are generally very large, heuristics are used to generate *good*, rather than optimal schedules. The search heuristic used in this paper is *best cost random depth first* (in addition to the heuristics about optimal starting times that are encoded in the model).

The stochastic analysis of our approach is done by discrete-event simulation using the performance evaluation tool MÖBIUS [19]. The simulation models are specified in the MODEST modeling language [13]. The approach presented in this paper does not depend on the usage of MODEST and MÖBIUS; any simulator with sufficiently expressive modeling formalism will do. Such simulators are rare, however. MODEST models, in particular, have a close correspondence to timed automata models, which makes it easy to keep both modeling worlds consistent.

The complete tool chain, together with the generated information, will be summarized in Section 4.4.

3.1 UPPAAL *and* UPPAAL CORA

Timed automata [3] are an extension of finite-state automata with real time clocks. Clocks can be reset, used in guards on transitions and in invariants on locations.

UPPAAL is an integrated tool environment for modeling, validation and model checking of real-time systems modeled as networks of timed automata, extended with data types (as bounded integers, arrays, etc.).

UPPAAL CORA [36] is a branch of UPPAAL for Cost Optimal Reachability Analysis. It can treat an extension of timed automata called linear priced timed automata (LPTA) [10,4]. LPTA allow one to annotate the model with the notion of cost. Costs can be accumulated with the delay while staying in a location, or as cost for a taking a particular transition. The model checker UPPAAL CORA then finds optimal paths matching goal conditions.

Scheduling synthesis by model checking [21,2,1] works along the following lines: Assume a model of the uncontrolled production process. The model checker searches for a reachable state with the property *all orders are finished*. Once it has found such a state the model checker can provide a diagnostic trace, *i.e.,* a trace from the initial state to the state with the desired property. This trace contains all the information on start and finish (times) of production steps, as well as information about resources, and, in the context of scheduling, represents a valid schedule.

This technique can also be applied to linear priced timed automata. Here, the model checker searches for the cheapest state reachable having the desired property. Having found such a state, the diagnostic trace then provides a cost-optimal schedule. In practice the model checker has to deal with very large state spaces which prohibits to find the cheapest state, respectively an optimal schedule. However, it has been shown that this technique, extended by suitable heuristics, allows to find *good* schedules in very short time [7,8].

What makes the timed automaton environment attractive for solving scheduling problems is the robustness against variations in the parameter setting. By this we mean that timed automata are a general description language and allow for an enormous variation in problem descriptions: it is easy to change, add, or remove model parameters without changing the search mechanism. On the other hand, this advantage co-

mes for the price of state-space explosion, which, however, can be handled by suitable heuristics.

3.2 MODEST *and* MÖBIUS

MODEST is a formal language to describe stochastic timed systems [18,13], equipped with a rigid formal semantics. The functional core of MODEST can be considered as a simple process algebra enriched with some convenient language constructs. The syntax resembles that of the programming language C and the modeling language Promela [26]. Data modularization concepts and exception handling mechanisms have been adopted from modern object-oriented programming languages such as Java. Process algebraic constructs have been strongly influenced by FSP (Finite State Processes [28]), a simple, elegant calculus that is aimed at educational purposes.

This core language is enriched with several modeling concepts tailored to model timed and stochastic systems. Important semantics concepts that are supported are (i) *Probabilistic branching* as a way to include quantitative information about the likelihood of choice alternatives; (ii) *Clocks* – like in timed automata – as a means to represent real time and to specify the dynamics of a model in relation to a certain time or time interval; (iii) *Random variables* as a means to give quantitative information about the likelihood of a certain event to happen after or within a certain time interval. The MODEST language allows one to specify *processes*, and to compose them in *parallel* using a parallel composition operator. Processes can manipulate *data variables* by *assignments*.

Processes can run in parallel, and can interact either by manipulating shared variables, or by synchronize over shared *actions*. The execution of actions within a process can be guarded by a 'when' clause, specifying an enabledness condition. In particular, the boolean expression in a 'when' clause may refer to clock values. In that case, an action becomes enabled as soon as the when condition becomes true.

MÖBIUS is a performance evaluation tool environment[2]. MÖBIUS supports multiple input formalisms and several evaluation approaches for these models. Atomic models are specified in one of the available input formalisms. Atomic models can be composed by means of state-variable sharing, yielding so-called composed models. Notably, atomic models specified in different formalisms can be composed in this way. This allows to specify different aspects of a system under evaluation in the most suitable formalism. Along with an atomic or composed model, the user specifies a reward model, which defines a reward structure on the overall model. Rewards are the vehicle to define the costs for our case study.

On top of a reward model, the tool provides support to define experiment series, called *Studies*, in which the user defines the set of input parameters for which the composed model should be evaluated. Each combination of input parameters defines a so-called *experiment*. Before analyzing the model experiments, a solution method has to be selected: MÖBIUS offers a powerful (distributed) discrete-event simulator, and, for Markovian models, explicit state-space generators and numerical solution algorithms. It is possible to analyze transient and steady-state reward models. The solver solves each experiment as specified in the *Study*. Results can be administered by means of a database.

MODEST has been integrated as an atomic modeling formalism into MÖBIUS by means of the tool MOTOR [15,14].

[2] The MÖBIUS software was developed by W.H. Sanders and the Performability Engineering Research Group (PERFORM) at the University of Illinois at Urbana-Champaign. See `http://www.mobius.uiuc.edu/`.

4 The Approach

In the EPT approach task durations are extended according to performance factors. These factors summarize the average time penalty incurred due to the failure behavior of resources. The problem with this approach is that schedules derived with EPT are actually not only scheduling resources, but, implicitly, also *downtimes* of resources. This is an adequate approach only in special cases, in particular, if the time to failure of resources and the respective repair times are likely to be short with low variance, compared to the length of processing times of tasks.

In the OST approach, the problem is tackled on a different level: instead of the processing times of individual tasks, complete recipes are considered. Due to the failure of resources, the run-time of an order is not of fixed length, but is rather of stochastic nature, which can ideally be described by a distribution function. The stochastic nature of resource failures makes it impossible to find a schedule that will always finish the order precisely at the due date. With some probability, the order is finished too early (in case that time has been reserved for the order, anticipating failure which did not occur), or too late (in case that the repair of resources took longer than anticipated). In both cases, costs are incurred. The case where an order will indeed finish in exactly the amount of time it was planned will be rare.

This observation indicates that it is fruitless to try to make every order cost-neutral. However, assuming the existence of a distribution function describing the run-time of an order makes it possible to reason about the *expected* costs incurred by an order. The goal of the OST approach is *to minimize the* expected costs *incurred by all orders*.

The approach works in two steps. First, optimal starting times (OSTs) for orders are derived, and second, schedules are generated which take these optimal starting times into account. To derive the OSTs, first the run-time distribution function of the different recipes *metal*, *uni* and *bronze* are estimated. This is done by simulating the three recipes on a machine model with breakdowns, and assuming a more-or-less accurate background load, but not the precise resource conflicts. From the order information, *i. e.,* due dates and cost factors, together with the respective estimated distribution, a time offset o from the respective due date d is derived such that the order, released at time $d-o$, incurs only minimal expected costs due to potential deadline misses. The time $d - o$ is the OST.

Using the OSTs, schedules are generated using UPPAAL CORA. The objective is to resolve resource conflicts between orders running in parallel, while keeping the starting times of the orders in the schedule as close as possible to the optimal starting time derived before.

In the following sections, the different aspects of the approach are described in more detail.

4.1 Optimal Starting Times

Given a distribution function $G(t)$ with density $g(t)$ describing the processing time T of a recipe, the OSTs are derived as follows. An order should start d_{opt} units of time before the due date, where d_{opt} is the offset from the due date which minimizes the mean cost incurred by the order. For a cost function $c(d,t)$ given by the values dcf and scf, the mean costs for an offset $\bar{d}$ is given by the integral

$$\int_0^\infty c(\bar{d},t)\cdot g(t)dt.$$

Thus, to calculate d_{opt}, we have to look for the zeros of

$$\frac{d}{dx}\int_0^\infty c(x,t)\cdot g(t)dt$$

Due to the piecewise linear shape of the considered cost functions, we can derive the following equation to characterize d_{opt}:

$$\frac{G(d_{opt})}{1-G(d_{opt})}=\frac{dcf}{scf}. \quad (1)$$

In the case that $G(t)$ is strictly increasing, d_{opt} is uniquely characterized. In case $G(t)$ is constant on a certain interval I, and Equation (1) holds on I, $d_{opt}=\inf I$ should be chosen. In case of discontinuities of G at point t such that $\frac{G(t')}{1-G(t')}<\frac{dcf}{scf}$ for $t'<t$ and $\frac{G(t')}{1-G(t')}>\frac{dcf}{scf}$ for $t'>t$ (or vice versa), $d_{opt}=t$ should be chosen.

It is thus straightforward to derive d_{opt} from $G(t)$, also when $G(t)$ is described by a discrete approximation.

Example. We assume an order with $dcf/scf=50$, and consider different distributions. For distribution G_{det} with

$$G_{det}(t)=0 \text{ for } t\in I_1=[0,300),$$

and

$$G_{det}(t)=1 \text{ for } t\in I_2=[300,\infty),$$

the fraction $\frac{G_{det}(t)}{1-G_{det}(t)}$ is constant 0 on I_1 and undefined on I_2. Although Equation (1) can not be satisfied formally, it is most reasonable to choose in this case $d_{opt}=300$, the position of the discontinuity of G_{det}.

As second example we consider Weibull distribution G_{Wei} with parameter $m=100$ and $k=2$ (*i.e.*, the distribution has a mean of 88.62, and the "firing" rate is increasing with time). $G_{Wei}(t)$ and $\frac{G_{Wei}(t)}{1-G_{Wei}(t)}$ are depicted in Figure 3. As can be seen, for $dcf/scf=50$, the solution to Equation 1 is rather pessimistic: $d_{opt}\approx 198.3$, which is more than twice the expected value of G_{Wei}.

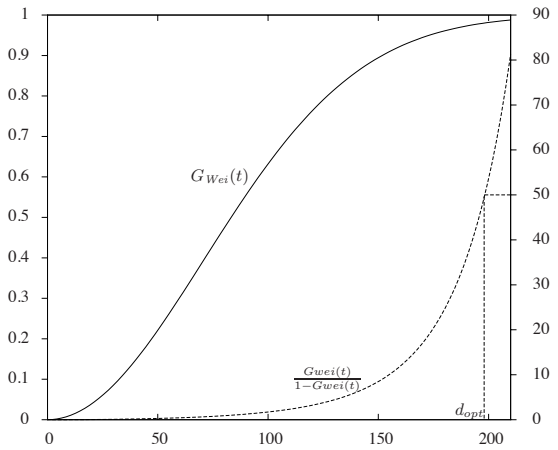

Fig. 3. Distribution $G_{Wei}(t)$ and $\frac{G_{Wei}(t)}{1-G_{Wei}(t)}$

4.2 Estimating Order Completion-Time Distributions

The principal approach to estimate distribution $G(t)$ is simulation. Orders, resources, and failure behavior are modeled in MODEST and simulated with MÖBIUS. To obtain meaningful results, it is necessary to model the environment in which an order usually runs. There are two main influences on the execution time of an order: first, the possible failures of resources, and second, the influence of other orders running in parallel. As far as the failures of resources go, we model the breakdown and repair behavior of all resources by means of two distributions, describing the *time-to-failure,* and the *time-to-repair,* respectively. In Section 5 we will describe in detail which distributions and which parameters for the distributions we have chosen.

To account for the mutual influence of orders running in parallel on the completion time of the order requires to create a reasonable background load in the simulation model. In essence, we created the following load model:

Parallelism The description of the case study limits the number of orders that can be executed in parallel to five.

Starting of orders We start five orders at random times, where the starting times are drawn according to a uniform distribution over an interval of length 720 minutes (12 hours). The completion time of one of these orders is measured.

Type of orders The type of the orders (metal, bronze, or uni), are chosen at random (each with equal probability), except for the one that is measured.

We assume that tasks are started as soon as they become ready and resources are available. Deliberate idling, unless specified in the recipe, is not allowed.

In our experiments that we describe in Section 5, we have carried out one million simulation runs, producing a histogram ranging from 0 to 20000 minutes with a step size of 10 minutes.

4.3 Schedule Generation

Schedules are synthesised with the model-checking tool UPPAAL CORA [36]. The production plant and the different orders are specified in terms of *priced timed-automata*.

First, we generate two sets of schedules, one based on net processing times, the other one based on the extended processing times. Accordingly, the first set of schedules tries to minimise the costs without taking the machine breakdown into account. The second one does take machine breakdowns into account following the EPT approach.

Second, we generate schedules based on the OSTs. Here, we have followed a different strategy: the objective was to find schedules with starting time as close as possible to the OSTs given. Taking simply the OSTs as starting times is not enough: there are still resource conflicts that have to be resolved, and these may require to start some orders earlier than the OST would suggest.

In the remainder of this section we want to sketch the timed automaton models used for scheduling. The models used here build on earlier work [7,8].

To represent the uncontrolled production process we have a timed automaton for each order. More precisely, each of the three recipes is modelled as a timed automaton with parameters (template), and instantiated these templates with earliest starting time and due date, which defines the orders.

We identified states which are "too late", *i.e.,* where the order is still processed although the due date has already passed, and assigned delay costs to them. Similarly, storage costs are assigned to states, where the order is already finished, but the due date has not been reached.

In order to reduce the state space it is necessary to add some heuristics, i.e. here heuristics reduce the serach tree and also the set of schedules that can be found. In earlier work [7,8] we experimented with several heuristics. In the current context, we use greediness as our only strategy, and, additionally, restrict the number of orders that may be processed at any point in time.

Note, that these heuristics are on the search level in the sense that they reduce the search space. The "full" search space contains all schedules based on the OST strategy, which is an heuristics on the modelling level.

The most efficient search strategy turns out to be *best cost random depth first*, meaning that among all direct successors of the current state having the same best cost, one successor is selected randomly for further search. The schedules we get in this way are *good*, rather than *optimal*.

For the first two sets of schedules, as mentioned above, one based on net processing times, the other one based on the extended processing times, we use the model with cost annotation as defined in the case description: storage and delay costs defined for each order individually.

For the OST based schedules we define a metric that reflects *as close as possible to the optimal starting time*. The intuition of this metric is that for orders with higher storage than delay costs, starting after the optimal starting time will be punished less, and for orders with higher delay than storage costs, starting earlier will be punished less. To implement this, we shift the cost definition for the completion time to the optimal starting time. The costs for the schedules synthesised then indicate how "good" the optimal starting times are approximated.

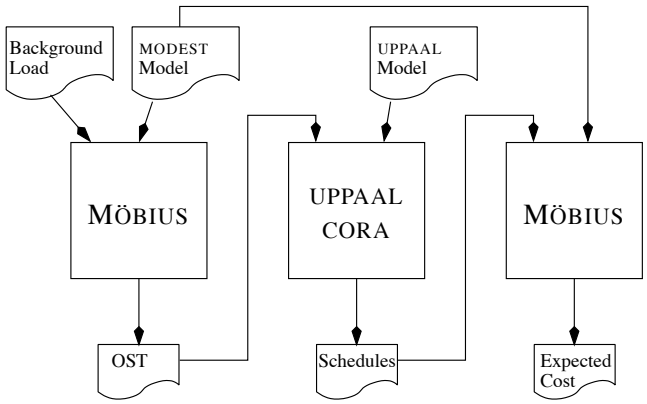

Fig. 4. Tool Chain

4.4 Summary

In Figure 4 the tool chain is depicted which was used to produce the results of this paper. A MODEST model of the plant, together with a background-load model, is simulated in MÖBIUS to obtain the OST. These, together with an UPPAAL model of the plant server as input to UPPAAL CORA, which generates schedules. The schedules are then added to the MODEST model of the plant and are simulated in MÖBIUS, resulting in the expected cost of executing the scheduled orders.

5 Experiments

5.1 Assumptions on Failure Behavior

In order to experiment with our approach to derive cost-optimal starting times (*cf.* Section 4.2), we need to make several assumptions on the failure behavior of the resources. In this section, we define four different sets of parameters which define the experiments, and describe the reasoning behind our choices.

In the original case description, the failure behavior of the resources is described only by the *performance factor* $p_r = \mathsf{uptime}/(\mathsf{uptime} + \mathsf{downtime})$ (*cf.* Section 2). The performance factor itself is stochastically quite meaningless, since it does not give any indication on the durations of up- and downtimes. Thus, we have to make assumptions on these times. As a first approach to relate up- and downtimes to durations, we introduce the so called *pace* parameter, which we define as $pace = E[\mathsf{uptime} + \mathsf{downtime}]$. We then define $E[\mathsf{uptime}] = p_r pace$ and $E[\mathsf{downtime}] = (1-p_r)pace$. The pace, together

Scenario	uptime	downtime	pace
E100	Exponential	Exponential	100 h
E31	Exponential	Exponential	31.6 h
WU100	Weibull	Uniform	100 h
WU31	Weibull	Uniform	31.6 h

Table 2. The considered failure behavior of resources

with the performance factors, gives information about the failure rate and the mean time between failure. The higher the pace, the higher the mean time to failure and the higher the mean repair time. For our study, we chose two different values of paces: 100 hours, and $10\sqrt{10} \approx 31.6$ hours.

For the purpose of simulation, it is necessary to describe uptime and downtime by means of distributions. The natural choice here is the exponential distribution: it is solely characterized by one parameter, the mean value, and is the only distribution with this property. It is therefore the choice with the *maximal entropy* (*cf.*, e.g., [23]).

In order to see how far the distributions of uptime and downtime influence our results, we consider a second set of distributions to use in our experiments. The very common distribution to describe times to failures is the Weibull distribution. A Weibull distribution is related to the exponential distribution, however, where the exponential distribution has a constant rate, the Weibull distribution has a rate variable over time, depending on one of its parameters. We chose deliberately a Weibull failure distribution where the failure rate of the component increases polynomially with degree 1.8 over time. This parameter is the only deliberate choice we make here. The other parameters of the Weibull distribution are determined by the previous assumptions on the pace and the performance factors. Together with the Weibull distribution for the time to failure, we choose for the time to repair an *uniform distribution*, which again is entirely determined by the pace and performance factors.

In comparison, the time to failure is in the exponential case more *random* than in the Weibull case: the squared coefficient of variation of an exponential distribution is 1, whereas for the Weibull distribution, as we use it here, the coefficient is 0.33. Therefore, the failure of a resource is more "predictable" than in the exponential case.

In summary, we consider in our study four different scenarios, depending on the choice of paces and distributions, as shown in Table 2. We name the four scenarios *E100*, *E31*, *WU100*, and *WU31* throughout the rest of the paper.

Simulating the OST for the different experiments up to certain precision of the histograms takes several hours. Prolonging the simulation time has not shown any significant change in the OSTs.

5.2 Generation of Optimal Starting Times

As described in Section 4.1, it is necessary to derive the completion time distribution for each recipe. We have done this

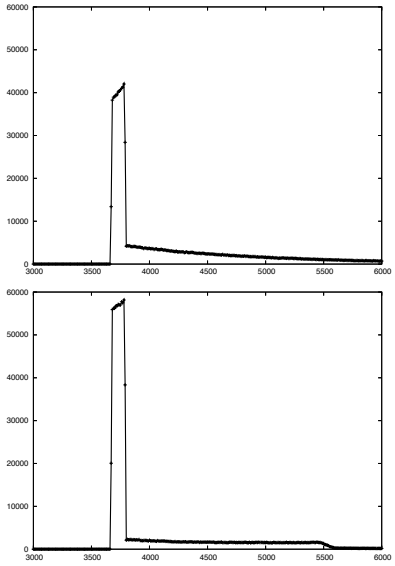

Fig. 5. Histogram for the cases E100 and WU100 for job type *metal*

for the four scenarios described in 5.1 by means of simulation. In Figure 5 we see the histogram for *metal* orders in the case of *E100* and *WU100*. Using Equation (1), it is straightforward to determine how long before its due date an order should be started. We call this the *optimal offset* of the order. We derive thus for each order an optimal offset. Subtracting these offset from the due date of the respective order yields the optimal starting time. In Table 3, we give the optimal offset and the OST for the case *E100* and job type *metal*. The recipe of job type *metal* in Figure 1 indicates that the minimal time that a *metal* order needs to complete equals 5037 minutes (the length of the critical path). As we can see in the table, the optimal offsets derived by our approach are in the range of 7660 to 7810 minutes, *i.e.*, about 2700 minutes (ca. 2 days) earlier. These 2700 minutes, approximately 35% of the entire time, is thus reserved for potential breakdowns of resources. In Table 5, we see the offsets and OSTs for *uni* orders. The *uni* recipe in Figure 1 shows that the minimal completion time of a *uni* job is about 6580 minutes (about 4.5 days). In Table 5, the optimal offsets are quite diverse. For orders 21, 23–25, and 28–29, the offsets are in the order of 19000 minutes ($\approx$ 13 days)—nearly three times the minimal time. On the other hand, for orders 22, and 26–27, the optimal offset lies in the area of 8000 minutes, which is only 20% longer than the minimal time. The reason for that can be found in Table 1: for orders 22 and 26–27, $scf > dcf$, which means that, if the due date is to be missed, it is preferable to be late, rather than early. Evidently, the optimal offsets for the respective orders are taking this into account.

5.3 Generation of schedules

Using UPPAAL CORA, we produced 6 sets of schedules:

– 10 Schedules, using net processing times. We call this class *S.NPT*.

Order	Offset	OST
1	7810	19634
2	7700	25648
3	7780	25856
4	7800	28572
5	7720	29372
6	7800	32700
7	7770	33929
8	7800	39372
9	7710	49254
10	7740	49512
11	7710	49829
12	7790	54358
13	7660	60104
14	7740	60240
15	7800	64068

Table 3. Optimal offsets and starting times for *metal* orders for *E100* in minutes

Order	Offset	OST
16	5410	38090
17	10750	39830
18	10850	49810
19	10810	51049
20	5400	59700

Table 4. Optimal offsets and starting times for *bronze* orders for *E100* in minutes

Order	Offset	OST
21	19390	11069
22	8320	33308
23	18920	24844
24	18710	25294
25	19070	31510
26	8110	52478
27	7930	53858
28	18810	58314
29	19070	66694

Table 5. Optimal offsets and starting times for *uni* orders for *E100* in minutes

- 10 Schedules, using extended processing times. We call this class *S.EPT*.
- Four classes of schedules with 10 schedules each, taking the optimal starting times of the four scenarios *E100*, *E31*, *WU100*, and *WU31* into account (*cf.* Section 5.1). We call these four classes of schedules *S.E100*, *S.E31*, *S.WU100*, and *S.WU31*, respectively.

The objective function for the *S.NPT* and *S.EPT* schedules was to meet the due dates as well as possible, causing costs for storage and delay as in the case description.

For the four classes *S.E100*, *S.E31*, *S.WU100*, and *S.WU31*, the objective function for the schedule synthesis was to find schedules that have the starting times of orders as close as possible to the OSTs and that have resolved the resource conflicts.

The search strategy we use is *best cost random depth first*. Due to the randomness and the size of the search space we get schedules of different quality, *i.e.,* with high cost variations. In order to generate a set of ten good schedules we perform 30–100 of schedule searches, depending on the model. A single experiment, however, takes less than a minute in most cases, less than 3 minutes in all cases.

The most efficient heuristic used is the limitation of the number of orders that are processed at each moment. The upper bound we choose is 5, which is the result of a number of experiments. This is plausible, considering the bottleneck resources: the dose spinners, that are available twice, but have to be used by every order, the dispersing line, and to some extent also the mixing vessels. If several orders are waiting for a resource, the the one who gets the resource is chosen nondeterministically, *i.e.,* there is no queuing.

5.4 Analysis by simulation

As explained in Section 4, our method to assess the quality of deterministic schedules is stochastic simulation. In this section we will present the results of the simulations.

In this experiment, we simulated seven classes of schedules. The first six classes are those generated with UPPAAL CORA, as described in the previous section. The seventh class is a set of four schedules, based on the OST only. Since we take in our approach only the starting times of orders into account, it is possible to regard the optimal starting times already as schedules as well: we start each order exactly at its optimal starting time. Thus, for each of the four schedule classes *S.E100*, *S.E31*, SWU100, and *S.WU31* we have one extra schedule, defined by the OSTs for that class. We call the respective schedule *S.OST.E100*, *S.OST.E31*, *S.OST.WU100*, and *S.OST.WU31*.

Our measure of interest is, for each schedule, the mean total cost that accumulates during the execution of all 29 orders. We simulated all schedules of all considered classes. The mean value is estimated with a relative confidence interval $< 10\%$ and a confidence level of 99%.

The schedules in class *S.NPT* and *S.EPT* were simulated for all the scenarios *E100*, *E31*, *WU100*, and *WU31*. The schedules *S.E100*, *S.E31*, SWU100, and *S.WU31* were each only simulated with the corresponding scenario, *i.e., S.E100* with *E100*, etc.

We present the average ($E[\cdot]$) and relative standard deviation (coefficient of variation c_v) of the costs of all schedules in the respective scenarios. These results are displayed in Table 6.

5.5 Interpretation of Simulation Results

5.5.1 Comparison of the *S.NPT* and *S.EPT* schedules

The average costs of the *S.NPT* schedules range from 8 million to 12 million, depending on the chosen scenario. The

Failure model used for schedule analysis

Schedule set	Measure	*E100*	*E31*	*WU100*	*WU31*
S.NPT	$E[\cdot]$	12.131.001	9.628.586	9.720.134	8.568.726
	c_v	11.88%	15.51%	14.16%	16.97%
S.EPT	$E[\cdot]$	5.907.007	3.340.511	3.800.883	2.530.444
	c_v	17.17%	29.28%	26%	37.12%
S.E100	$E[\cdot]$	1.683.969			
	c_v	1.93%			
S.E31	$E[\cdot]$		855.446		
	c_v		3.76%		
S.WU100	$E[\cdot]$			1.313.474	
	c_v			4.79%	
S.WU31	$E[\cdot]$				785.304
	c_v				7.47%
S.OST.E100		1.732.719			
S.OST.E31			1.445.109		
S.OST.WU100				2.209.187	
S.OST.WU31					2.165.868

Table 6. Results for total cost for different schedules

relative standard deviation ranges from ca. 12 to 17%. This shows that, even though all schedules in this class resolve resource conflicts, the way it is done has an influence on the costs that are incurred.

The average costs of the *S.NPT* schedules are the highest in the whole table. This is not surprising, since these schedules do not take the failure of resource into account *at all*. Therefore, for each order the probability to overshoot the due date by a considerable amount of time is very high. The *S.NPT* numbers show how bad costs can become.

The average cost of schedules in the *S.EPT* class range from ca. 2.5 million to 6 million, depending on the chosen failure scenario. The relative standard deviation is between 17% and 38%. The *S.EPT* schedules are better than the *S.NPT* schedules. We explain this by the fact that more time is calculated for the execution of each task, and thus for each order. The numbers suggest that *S.EPT* can indeed help saving considerable costs, compared to the *S.NPT* schedules.

The variation in the costs for the *S.NPT* and *S.EPT* schedules confirm that there are good schedules and bad schedules. At this point we are not able to determine *a priori*, what schedule is a good one without simulation.

5.5.2 Comparison of the *S.EPT* and the *S.E*/*S.WU* schedules

We consider first the *S.E100* case. Table 6 shows, that the average costs incurred by these schedules is about 1.7 million The relative standard deviation is 1.93%, which is very low. We can see, that the average cost of the *S.E100* schedules is only 28.5% of the average cost of the *S.EPT* schedules for the *E100* scenario. Since the variation of the *S.EPT* schedules is in the order of 17%, we can assume a best-case schedule which is indeed 17% better than the average. On the other hand, we can assume that there is a *S.E100* schedule that is 1.93% worse than the average. Setting best-case and worst-case schedules in relation, the costs of the worst-case *S.E100* schedule would still be only 35% of the best-case *S.EPT* schedule.

	E100	*E31*	*WU100*	*WU31*
S.EPT	5.907.007	3.340.511	3.800.883	2.530.444
S.E100	**1.683.969**	1.105.239	1.139.555	1.079.691
S.E31	2.232.010	**855.446**	986.859	702.696
S.WU100	3.047.240	1.070.889	**1.313.474**	684.322
S.WU31	3.842.751	1.416.179	1.769.831	**785.304**

Table 7. Average mean total costs for all OST schedule with all scenarios

For the *S.E31* schedules, the average costs are about 0.86 million, with a relative standard deviation of 3.8%. This is 25.6% of the costs of the *S.EPT* schedules in the *E31* scenario. Assuming again best-case/worst-case schedules, the *S.E31* schedule would still cost only 38% of the *S.EPT* schedule.

The *S.WU100* schedules reduce the costs to 34.5% of the *S.EPT* schedules, and, assuming the best-case/worst-case again, the cost is still reduced to about 49%.

Finally, the *S.WU31* cost on average only 31% of the *S.EPT* schedules, and in the extreme case, still only 53%.

These results lead us to the main conclusion of this paper: OST schedules are in general cheaper than EPT schedules, and thus better.

5.5.3 Comparison of *S.E*/*S.WU* and *S.OST.E*/*S.OST.WU* schedules

The schedules in the *S.E*/*S.WU* classes have been generated with UPPAAL CORA. Since the optimal starting times can also be seen as schedules themselves, the question arises whether UPPAAL CORA is needed at all, and if these *S.OST.E* and *S.OST.WU* schedules are not actually sufficient.

The numbers in Table 6 show that this is in general not the case. Even though for the *S.E100*/*S.OST.E100*, the costs are very close (around 1.7 million), for the other cases the differences become more pronounced: the UPPAAL schedules generate 60%, in the *WU31* case even only 36% of the costs of the corresponding OST schedules.

Our explanation for that is that the schedules generated with UPPAAL CORA do resolve resource conflicts by letting orders start earlier (if $scf < dcf$), or later (if $scf > dcf$). Thus, the effect is that the probability to complete the order on the expensive side of the cost function is reduced. Consequently, the step of producing refined OST with UPPAAL CORA is indeed necessary to reduce the mean total costs further.

5.6 Robustness of the OST approach

In this section we show that even if we make inaccurate assumptions on the failure distributions, and thus work with inaccurate OSTs to generate schedules, in all cases the costs are not higher than for the EPT approach, and in most cases significantly better. In order to find out how stable our approach is, we have simulated all *S.E*/*S.WU* schedules also with the parameters for which the schedules where *not* generated for.

We would expect that the simulations with the "proper" parameters should always give lower costs than the others. This is however not always the case. In Table 7, we see the averages for all combinations of schedule classes and failure scenarios. For comparison, we have also repeated the values for the *S.EPT* schedules from Table 6. The values set in boldface come from the same Table. For the case of scenario *E100* we see that the schedules in class *S.E100* produce minimal cost: the non-diagonal values in the respective column are all much higher. The same holds still for the case *E31*: the schedules of class *S.E31* are the lowest in the column.

The situation is different for the *WU* scenarios. In both cases, the corresponding *S.WU* schedules do not yield the lowest cost. In case of *WU100* it would be better to use a schedule from the *S.E31* class. In case of *WU31*, it would be better to use one from *S.WU100*.

The cause of this phenomenon is subject to further research. Nevertheless, even though wrong assumptions on distributions might give not the best results, a comparison with the *S.EPT* results shows that our OST approach produces still substantially better results.

6 Conclusions

In this article, we present an approach for proactive schedule generation for manufacturing scenarios with resource competition, breakdowns, and earliness/tardiness penalties. The pivotal elements of our approach are the so-called optimal start times of orders. These OSTs are estimated by means of simulation and straightforward mathematical derivations. They reflect release times of orders for which the expected cost incurred by the orders are minimal. We use the OSTs as input to our schedule synthesizer, the model-checker UPPAAL CORA.

We assess the properties of all generated schedules by means of stochastic simulation. The simulation models, described in the modelling language MODEST, are simulated with MÖBIUS. The simulation models comprise resources prone to failure, the recipes, and the schedule in question. We have considered four different cases of failure behavior for the estimation of OSTs, generation of schedules and the simulation of schedules.

Our main question was whether the approach using the OST as starting times gives better schedules than the EPT approach based on adding slack times, which is widely used in practice. The simulation results show that the schedules that we have derived with the OST approach are in all four scenarios considered substantially cheaper than the schedules derived with the EPT approach. Moreover, we showed that our approach is robust in the sense that, even if assumptions on the failure behavior of resources are inaccurate, the OST schedules still incur lower costs than their EPT counterparts.

Using timed automata and model checking techniques to derive schedules has been shown as a promising approach in cases where standard scheduling algorithms are not applicable, in a number of case studies. Here, we extend the tool set consisting of model checking techniques by stochastic simulation for both, improved input values, and also for evaluation of the results, justifying our initial claim.

Relevant questions that still remain open are what the right assumptions for derivation of the OSTs are, *e.g.,* what are the best estimates for distribution of resource failure, and what are the best assumptions on the background load of a system. Then, taking OSTs as a basis for derivation of schedules, the question which schedule approximates the OSTs best. We suggested a cost function with an obvious intuition, but possibly there are better strategies. Another direction could be to be more explicit with respect to the load in a system. There are moments with low load, and others where the load is high. Our OSTs are all based on high load, for moments with low load they are too pessimistic. We could calculate different OSTs for different loads and apply the best fitting OST during scheduling when it is clear how many concurrent orders have to be processed in the close future.

The scheduling strategy we followed in both schedule synthesis and simulation is greedy, and when there are several tasks waiting for a resource, one task is chosen nondeterministically to obtain the resource. Possibly other policies could give better results.

Because we have a general purpose modelling and analysis tool, we can easily model a great range of different settings, which is certainly one of the strong points of model checking. Yet, when the complexity of a problem is high, also model checking cannot change that in principle. In the presented case study, however, one might wonder to what extent it does make sense to try to solve a scheduling problem that has only very few "good" solutions in the context of possible machine failures - the chance of following a "good" schedule then is very low.

The tools we used, UPPAAL CORA and MÖBIUS have the necessary modelling and processing capabilities that are needed for the demonstration of our approach. However, these tools do not yet have industrial strength. Modelling effort that, in principle, can be automated was done by hand here, which is a time-consuming process. For an industrial application, modelling should be mechanized and performed with tool-support. Nevertheless, UPPAAL CORA and MÖBIUS are under continuous improvement, aiming at industrial applicability.

Acknowledgment

We give our thanks to Ed Brinksma for discussion on the approach, and to Martijn Hendriks and Gerd Behrmann, who helped us with models and tools.

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: Scheduling with timed automata. *Theor. comp. sci.* **354**(2) (2006) 272–300
2. Abdeddaïm, Y., Maler, O.: Job-shop scheduling using timed automata. In Berry, G., Comon, H., Finkel, A. (eds.): *Computer aided verification. LNCS*, Vol. 2102. Springer, Berlin (2001) 478–492
3. Alur, R., Dill, D. L.: A theory of timed automata. *Theor. comp. sci.* **126**(2) (1994) 183–235
4. Alur, R., La Torre, S., Pappas, G. J.: Optimal paths in weighted timed automata. In Di Benedetto, M. D., Sangiovanni-Vincentelli, A. (eds.): *Hybrid systems: computation and control; ... HSCC. LNCS*, Vol. 2034. Springer, Berlin (2001) 49–62
5. AMETIST, IST project ist-2001-35304 `http://ametist.cs.utwente.nl/`.
6. Amnell, T., Behrmann, G., Bengtsson, J., D'Argenio, P. R., David, A., Fehnker, A., Hune, T., Jeannet, B., Larsen, K. G., Möller, M. O., Pettersson, P., Weise, C., Yi, W.: UPPAAL: now, next, and future. In Cassez, F., Jard, C., Rozoy, B., Ryan, M. D. (eds.): *Modeling and verification of parallel processes: ... MOVEP. LNCS*, Vol. 2067. Springer, Berlin (2001) 99–124
7. Behrmann, G., Brinksma, E., Hendriks, M., Mader, A.: Production scheduling by reachability analysis: a case study. In: *Proceedings of the 19th IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE Computer Society (2005) 140a
8. Behrmann, G., Brinksma, E., Hendriks, M., Mader, A.: Scheduling lacquer production by reachability analysis: a case study. In: *Proceedings of the 16th IFAC world congress*. Elsevier (2005)
9. Behrmann, G., David, A., Larsen, K. G.: A tutorial on UPPAAL. In Bernardo, M., Corradini, F. (eds.): *Formal methods for the design of real-time systems: ... SFM-RT. LNCS*, Vol. 3185. Springer, Berlin (September 2004) 200–236
10. Behrmann, G., Fehnker, A., Hune, T., Larsen, K. G., Petterson, P., Romijn, J.: Guiding and cost-optimality in UPPAAL. In Khatib, L., Pecheur, C. (eds.): *Model-based validation of intelligence*. AAAI, Menlo Park (2001) 66–74
11. Behrmann, G., Larsen, K. G., Rasmussen, J. I.: Optimal scheduling using priced timed automata. *Perform. eval. rev.* **32**(4) (2005) 34–40
12. Bohnenkamp, H. C., Hermanns, H., Klaren, R., Mader, A., Usenko, Y. S.: Synthesis and stochastic assessment of schedules for lacquer production. In: *QEST '04*. IEEE Computer Society, Los Alamitos (2004) 28–37
13. Bohnenkamp, H., D'Argenio, P. R., Hermanns, H., Katoen, J.-P.: MODEST: a compositional modeling formalism for hard and softly timed systems. *IEEE trans. softw. eng.* **32**(10) (October 2006) 812–830
14. Bohnenkamp, H., Hermanns, H., Katoen, J.-P.: MOTOR: the MODEST tool environment. In Grumberg, O., Huth, M. (eds.): *Tools and algorithms for the construction and analysis of systems: ... TACAS. LNCS*, Vol. 4424. Springer, Berlin (2007) 500–504
15. Bohnenkamp, H., Hermanns, H., Katoen, J.-P., Klaren, R.: The MODEST modeling tool and its implementation. In Kemper, P., Sanders, W. H. (eds.): *Computer performance evaluation: ... TOOLS. LNCS*, Vol. 2794. Springer, Berlin (2003) 116–133
16. Butazzo, G. C.: *Hard real-time computing systems: predictable scheduling algorithms and applications*. Kluwer, Boston (1997)
17. Cai, X., Zhou, X.: Stochastic scheduling with asymmetric earliness and tardiness penalties under random machine breakdowns. *Probab. eng. inf. sci.* **20**(4) (2006) 635–654
18. D'Argenio, P. R., Hermanns, H., Katoen, J.-P., Klaren, R.: MoDeST: a modelling and description language for stochastic timed systems. In de Alfaro, L., Gilmore, S. (eds.): *Process algebra and probabilistic methods: ... PAPM–PROBMIV. LNCS*, Vol. 2165. Springer, Berlin (2001) 87–104
19. Deavours, D. D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J. M., Sanders, W. H., Webster, P. G.: The Möbius framework and its implementation. *IEEE trans. softw. eng.* **28**(10) (2002) 956–969
20. Demeulemeester, E. L., Herroelen, W. S.: *Project scheduling: a research handbook*. Kluwer, Boston (2002)
21. Fehnker, A.: Scheduling a steel plant with timed automata. In: *Sixth international conference on real-time computing systems and applications: RTCSA*. IEEE Computer Society, Los Alamitos (1999) 280–286
22. Gao, H.: Building robust schedules using temporal protection: an empirical study of constrained based scheduling under machine failure uncertainty. Master's thesis, Department of Industrial Engineering, University of Toronto (1995)
23. Harrison, P. G., Patel, N. M.: *Performance modelling of communication networks and computer architectures*. Addison-Wesley, Wokingham (1993)
24. Herroelen, W., Leus, R.: Robust and reactive project scheduling: a review and classification of procedures. *Int. j. prod. res.* **42**(8) (2004) 1599–1620
25. Herroelen, W., Leus, R.: Project scheduling under uncertainty: survey and research potentials. *European j. oper. res.* **165**(2) (2005) 289–306
26. Holzmann, G. J.: *The SPIN model checker: primer and reference manual*. Addison-Wesley, Boston (2004)
27. Lambrechts, O., Demeulemeester, E., Herroelen, W.: A tabu search procedure for developing robust predictive project schedules. *Int. j. prod. econ.* **111**(2) (2008) 493–508
28. Magee, J., Kramer, J.: *Concurrency: state models and Java programs*. Wiley, Chichester (1999)
29. Mehta, S. V., Uzsoy, R.: Predictable scheduling of a single machine subject to breakdowns. *Int. j. comput. integr. manuf.* **12**(1) (1999) 15–38
30. Mehta, S. V., Uzsoy, R. M.: Predictable scheduling of a job shop subject to breakdowns. *IEEE trans. robot. autom.* **14**(3) (1998) 365–378
31. MODEST site `http://www.purl.com/net/modest`.
32. Panek, S., Engell, S., Lessner, C.: Scheduling of a pipeless multi-product batch plant using mixed-integer programming combined with heuristics. In Puigjaner, L., Espuña, A. (eds.): *European symposium on computer-aided process engineering, 15: ... ESCAPE-15. Computer-aided chemical engineering*, Vol. 20 A/B. Elsevier, Amsterdam (2005) 1033–1038
33. Pinedo, M. L.: *Planning and scheduling in manufacturing and services*. Springer series in operations research. Springer, New York (2005)
34. Stork, F.: *Stochastic resource-constrained project scheduling*. PhD thesis, Technical University, Berlin (2001)
35. UPPAAL home page `http://www.uppaal.com`.
36. UPPAAL CORA site `http://www.cs.aau.dk/~behrmann/cora`.