Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 296 (2013) 163–181

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Extending a Synthesis-Centric Model-Based Systems Engineering Framework with Stochastic Model Checking

J. Markovski[1,2] E.S. Estens Musa[3] M.A. Reniers[4]

Department of Mechanical Engineering,
Eindhoven University of Technology,
Eindhoven, The Netherlands

Abstract

We propose to integrate performance evaluation with supervisory control synthesis to bring higher confidence in the control design. Supervisory control theory deals with automatic synthesis of supervisory controllers that ensure safe behavior of the supervised system, based on the models of the uncontrolled system and the (safety) control requirements. For the purpose of performance evaluation, we turn to stochastic model checking of continuous-time Markov chains, which requires an extension of the model of the uncontrolled system with Markovian delays. We cast our proposal as an extension of a model-based systems engineering framework that relies on supervisor synthesis. We treat the Markovian delays syntactically, exploiting their equivalent interleaving behavior with uniquely-named uncontrollable transitions. In this way, we can employ already available synthesis tools, while preserving the stochastic behavior. To this end, we develop model transformation tools to extract the underlying Markov process from the stochastic discrete-event model of the supervised system. We illustrate the approach by modeling a pipeless plant that employs automated guided vehicles instead of fixed piping in order to ensure greater flexibility of the plant. The control problem that we solve is safe high-level movement coordination of the vehicles, ensured by the supervisory controller. We show how to seamlessly introduce stochastic behavior in the supervised system and we evaluate several performance and reliability aspects of the plant. We implement the framework by interfacing two state-of-the-art tools: Supremica for supervisory controller synthesis and MRMC for Markovian model checking. To this end, we improve previous attempts by providing support for data-based observers, which greatly improve the modeling capabilities of the framework.

Keywords: supervisory control theory, performance evaluation, Markov processes

1 Introduction

The constant increase of complexity of high-tech complex systems is taking its toll on the development of control software. Adaptations of traditional software development approaches that employ (re)design-(re)coding-testing loops have proven not

[1] J. Markovski is supported by Dutch NWO project: ProThOS, no. 600.065.120.11N124.

[2] Email: j.markovski@tue.nl

[3] Email: eestens@gmail.com

[4] Email: m.a.reniers@tue.nl

1571-0661 © 2013 Elsevier B.V. Open access under CC BY-NC-ND license.
http://dx.doi.org/10.1016/j.entcs.2013.07.011

entirely adequate to handle the challenge. Namely, the specifications of the control requirements that the software must implement frequently change during the design process inducing a large number of (time-consuming) recoding iterations [18]. This issue becomes even more prominent as modern markets impose increasing demands for better quality, performance, safety, and ease of use, giving rise to (formal) approaches for generation of control software.

Supervisory Control

Supervisory control theory [29,6] proposes a particular solution for control of discrete-event systems by investigating methodologies for automatic synthesis of models of supervisory controllers, referred to as *supervisors*. Supervisory controllers coordinate high-level system behavior by observing discrete(-event) system behavior, e.g., by receiving sensor signals from ongoing activities, making a decision on allowed activities, and sending back feedback to the system, guiding its safe execution, e.g., by sending control signals to the hardware actuators. The supervisors are synthesized based on formal models of the uncontrolled hardware, referred to as *plant*, and the model of the *control requirements*. Thereafter, supervisory control software can be generated based on these models.

The observed activities of the system are modeled by means of discrete events, which are split into *controllable events*, which can be disabled by the supervisor in order to prevent unsafe or otherwise undesired behavior, and *uncontrollable events*, which the supervisor can only observe in order to make a correct control decision. Controllable events typically model activities like interaction with the actuators of the machine, whereas uncontrollable events model activities like observation of sensors or user interaction over which the system has no control.

The synchronization of the plant and a supervisor, referred to as *supervised plant*, models the supervisory control loop, i.e., the coupling of the unsupervised system and the controller. A standard assumption is that the controller reacts sufficiently fast on machine input, which enables the modeling of the supervisory control feedback loop as a pair of synchronizing processes [29,6]. The supervisor can disable controllable events by not synchronizing with them, but it must always enable available uncontrollable events by synchronizing with them as the latter cannot be ignored. In addition, supervised plants must satisfy the control requirements, which model allowed or safe system behavior.

Need for Performance and Reliability Evaluation

The synthesized supervisors guarantee only safe functioning of the system as prescribed by the (safety) control requirements. In addition, they also typically ensure nonblocking behavior [29,6], i.e., they prevent deadlock and livelock. In order to ensure the latter, the plant is augmented with so-called marked states, which model situations in which the system can successfully terminate its execution. It is required that a marked state can be reached from each state of the system, thus offering an option for successful termination, which brings some level of confidence

in the system design.

In order to ascertain safety properties, the synthesis procedure may require elimination of important states that denote desired progress of the system. This situation occurs when there exists a trace from such a state that may lead the execution of the system to unsafe situations. In most cases, the latter is not directly deducible from the control requirements and, moreover, it can only be observed during or following the synthesis procedure. If such a situation occurs, then either the control requirements are too strict, or the model of the uncontrolled system is not sufficiently detailed or it is flawed, i.e., wrongful assumptions have been made. Moreover, in addition to the possible absence of progress properties, even if the system is validated to perform functionally as intended, there are absolutely no guarantees about its performance or reliability. Therefore, in order to establish presence or absence of such performance or reliability properties, additional quantitative analysis must be performed.

For a broader discussion on ensuring correct functionality of the supervised system, we refer to [22,26]. In the setting of this paper, we discuss the integration of performance evaluation in the model-based system engineering framework of [25]. We use Continuous Stochastic Logic (CSL) [4] for expressing performance requirements and use the Markov Reward Model Checker (MRMC) [14] for establishing validity of these properties.

There is also related work, extending supervisory control with quantitative aspects like probabilities [17,10,28] and stochastic delays [15,16], in order to ascertain that extra performance or reliability requirements are met as well. Most of these approaches treat controllability and optimality at the same time. With respect to optimality, the supervisor must ensure optimal behavior with respect to a given set of performance measures. The problem of optimality has also been tackled in the field of performance evaluation, employing the wide-spread class of Markov decision processes [13]. The control problem is to schedule the control actions such that some performance measure is optimized. Stochastic games problem variants [7] and specifications of control strategies using probabilistic extensions of temporal logics [2,5,8] are also emerging in the formal methods community.

Outline and Contributions

Unlike other approaches, we treat the stochastic delays syntactically and we decouple the synthesis procedure from ensuring optimality. We restrict to Markovian (exponentially-distributed) delays with which the plant is extended, since we employ the memoryless property to enable a syntactic compositional treatment. This approach enables us to handle industrial systems, by employing state-of-the-art tools Supremica [1] for efficient supervisor synthesis and MRMC [14] for performance evaluation by means of stochastic model checking. The syntactical treatment of Markovian delays is enabled by the fact that uniquely-named uncontrollable events and Markovian delays have the same interleaving behavior [25]. Thus, (1) if we replace Markovian delays with uncontrollable events in the plant, ensuring uniqueness of labels, then (2) perform supervisor synthesis on the renamed plant, which

preserves the uncontrollable events, and finally, (3) rename back to the original Markovian delays to obtain the stochastic supervised plant, and extract the underlying Markov process, we have a procedure for supervisor synthesis of stochastic discrete-event system and derivation of a performance model that can be fed to the stochastic model checker MRMC. Step (1) is performed manually for step (2) we employ Supremica, and for step (3) we develop a transformation tool referred to as Supremica2MRMC [24], which extends the algorithm of [12].

Extending our previous work [25], in the setting of this paper, we provide for data-based control requirements, relying on the process theory of [23]. To this end, we employ *data-based observers*, which provide additional data-based information to the supervisor, by observing important sequences of events in the plant. Observers are discrete-event processes that present an augmentation of the plant and that detect important states of the plant based on the history of observed events. In this way, this important information can be directly communicated to the supervisor and employed for supervision. The need for observers becomes evident when the plant is event-based, whereas the control requirements are more naturally expressed as state- or data-based properties. Note that the plant models only the unsupervised behavior of the system, so any additional information must be generated by the observers. To enable the use of data-based observations, we have to extend the translation tool Supremica2MRMC appropriately.

We illustrate the proposed framework on a case study involving coordination of movement of automated guided vehicles of a pipeless plant. Pipeless plants are an alternative to the traditional recipe-driven multipurpose batch plants, which replace the pipes with automated guided vehicles that carry containers [19]. We discuss the modeling process, the coordination requirements, the obtention of a performance model of the pipeless plant, and the process of formalization and verification of several performance properties.

2 Synthesis-Centric Systems Engineering

For supervisor synthesis we rely on the tool Supremica [1] that employs extended finite automata [32] to model the plant and the control requirements. Extended finite automata are extensions of finite nondeterministic automata with state labels, data assignments, and guarded transitions. We rely on data-based control requirements, where specific states of the system will be identified by the variable assignments. We employ two prominent forms of control requirements: state exclusion, which specifies which combination of states of the concurrent components of the plant are allowed, and state-transition exclusion, which specifies which events are allowed in a given observation of states. Ideally, we should be able to refer directly to states, employing the state labels, but the current release of the synthesis tool [1] does not yet support such requirements, so we rely on variables to identify states of interest.

Extended Finite Automata

The set of data elements is given by D, where only finite integer and enumerated types are currently supported by the synthesis tool [1]. The set of data variables is denoted by V, and by X we denote data expressions involving standard arithmetical operations [1] evaluated with respect to $e_D\colon \mathsf{X} \to \mathsf{D}$ for $D \in \mathsf{D}^{\mathsf{V}}$. The guarded commands are given as Boolean formulas, where the logical operators are given by $\{!, \&, |, =>\}$, denoting negation, conjunction, disjunction, and implication, respectively, and the atomic propositions are formed by the predicates from the set $\{<, ==, !=, >\}$, denoting the relations smaller, equal, nonequal, and larger, respectively, between data variables and data elements. We use B to denote the obtained Boolean expressions, evaluated with respect to a given valuation $v_D\colon \mathsf{B} \to \{\mathrm{ff}, \mathrm{tt}\}$ for $D \in \mathsf{D}^{\mathsf{V}}$, where ff denotes the logical value false, and tt denotes true. The set of event labels is given by E, whereas states are labeled by elements from the set S.

An extended finite automaton A can be represented by a tuple $A = (\mathsf{S}, \mathsf{E}, \mathsf{V}, \longrightarrow, s_0, v_0, M)$, where $s_0 \in \mathsf{S}$ is the initial state of the automaton, $v_0 \in \mathsf{D}^{\mathsf{V}}$ is the initial assignment of the variables, and $M \subseteq \mathsf{S}$ is the set of marked states. The guarded labeled transition relation is represented as $\ell \xrightarrow{g:e:u} \ell'$, where $\ell, \ell' \in \mathsf{S}$ are the locations, given by the state labels, $g \in \mathsf{B}$ is the guard, $e \in \mathsf{E}$ is the event label, and $u\colon \mathsf{D}^{\mathsf{V}} \to \mathsf{X}^{\mathsf{V}}$ is the variable update function. The underlying discrete-event model are standard finite automata in which the states are formed by the original state labels coupled with the variable assignments, whereas the labeled transition is enabled if the guard is evaluated to true. In the resulting state, the variables are subsequently updated, as given by the operational rule

$$\frac{\ell \xrightarrow{g:e:u} \ell',\ v_d(g) = \mathrm{tt}}{(\ell, d) \xrightarrow{e} (\ell', e_d(u(d)))},$$

where the initial state of the automaton is given by (s_0, v_0).

Extended finite automata are coupled using the automata synchronous parallel composition of [27], which synchronizes on the events in the common alphabet of both automata, whereas it interleaves on the rest. To define the composition, we need an auxiliary operation $f|_D$, which denotes the restriction of the function $f\colon A \to B$ on the domain $D \subseteq A$. Suppose that we have two automata $A_1 = (\mathsf{S}_1, \mathsf{E}_1, \mathsf{V}_1, \longrightarrow_1, s_{10}, v_{10}, M_1)$ and $A_2 = (\mathsf{S}_2, \mathsf{E}_2, \mathsf{V}_2, \longrightarrow_2, s_{20}, v_{20}, M_2)$. Their synchronous parallel composition is given by the automaton $A_1 \parallel A_2 = (\mathsf{S}_1 \times \mathsf{S}_2, \mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{V}_1 \cup \mathsf{V}_2, \longrightarrow, (s_{10}, s_{20}), v_{10} \cup v_{20}|_{\mathsf{V}_2 \setminus \mathsf{V}_1}, M_1 \times M_2)$, where $v_{10}|_{\mathsf{V}_1 \cap \mathsf{V}_2} = v_{20}|_{\mathsf{V}_1 \cap \mathsf{V}_2}$. Now, suppose that $\ell_1 \xrightarrow{g_1:e_1:u_1} \ell_1'$ and $\ell_2 \xrightarrow{g_2:e_2:u_2} \ell_2'$. Then, the available transitions of (ℓ_1, ℓ_2) are given by:

(i) $(\ell_1, \ell_2) \xrightarrow{g:e:u} (\ell_1', \ell_2')$, if $e_1 = e_2 = e$ and $u_1|_{\mathsf{V}_1 \cap \mathsf{V}_2} = u_2|_{\mathsf{V}_1 \cap \mathsf{V}_2}$, where $g = g_1 \& g_2$ and $u = u_1 \cup u_2|_{\mathsf{V}_2 \setminus \mathsf{V}_1}$.

(ii) $(\ell_1, \ell_2) \xrightarrow{g_1:e_1:u_1} (\ell_1', \ell_2)$, if $e_1 \in \mathsf{E}_1 \setminus \mathsf{E}_2$.

(iii) $(\ell_1, \ell_2) \xrightarrow{g_2:e_2:u_2} (\ell_1, \ell_2')$, if $e_2 \in \mathsf{E}_2 \setminus \mathsf{E}_1$.

The synchronous parallel composition forces synchronization of events on the common alphabet, provided that the updating of the variables coincides, whereas it

allows interleaving for the other events. Synchronous parallel composition is both commutative and transitive.

Model-Based Systems Engineering Framework

Our proposal for application of supervisory control theory in development of high-tech machines has been cast as a model-based systems engineering framework [30,26]. Synthesis-centric model-based approaches to systems engineering enable rapid prototyping as one can couple the models with (prototype) hardware components to evaluate the control requirements, before building and testing expensive control software. Initially, the domain engineers make the desired system specification, and contrive it into a design together with the software engineers. The architectural design defines the modeling level of abstraction and control architecture resulting in informal specifications of the plant, and the model of the control requirements. For synthesis purposes, any continuous behavior must be abstracted from in the plant, resulting in a discrete-event model. In parallel, the control requirements are modeled, and together with the plant, serve as ingredients for the supervisor synthesis.

Following the supervisor synthesis, the control must be validated as meaningful, i.e., desired functionalities of the controlled plant must be preserved to ensure that the system and the control requirements have been correctly modeled. If validation fails, remodeling of the control requirements and the plant, or even complete revision may prove necessary. Finally, the control software is generated automatically from the validated models, shifting the focus of software engineers from coding to modeling.

Derivation of a Performance Model

We model stochastic behavior by means of Markovian (or exponentially-distributed) delays that are introduced orthogonally in the extended finite automata in the vein of the Interactive Markov chains of [11]. In such a stochastic extension, the Markovian delays are simply interleaved in the parallel composition, which preserves the stochastic behavior of the synchronization. Having in mind the definition of the synchronous parallel composition from above, if we treat Markovian delays as uniquely-named (uncontrollable) delays, then we preserve their interleaving behavior [25]. To show that controllability is preserved as well, we have to extend the notion of controllability appropriately. This exercise has been done in [25], proving that the transformation does not alter the stochastic behavior of the supervised plant.

To enable performance evaluation and reliability analysis, we need to extract a performance model from the supervised plant that comprises all relevant stochastic information. To this end, we augment the model-based framework as shown in Fig. 1. The changes to the framework are in the synthesis and the post-synthesis (verification) phase. The modeler specifies the plant, renaming Markovian delays and recording the relation between the uncontrollable events and the Markovian delays in a file with extension .u2m. Following the supervisor synthesis, based on the

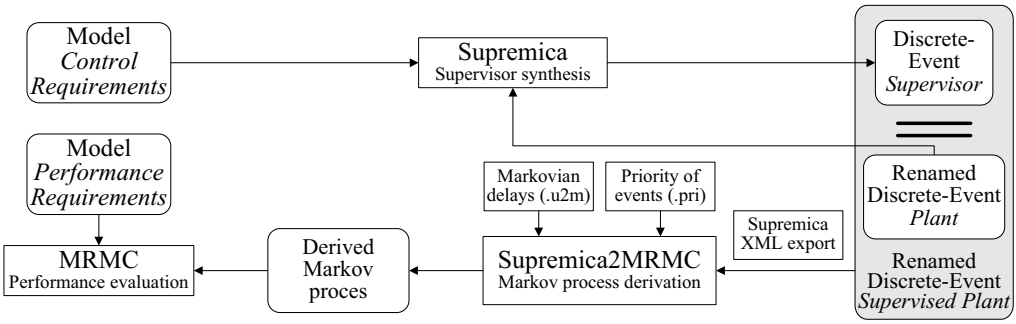

Fig. 1. Extension of the systems engineering framework with stochastic model checking and the accompanying tools

(discrete-event) plant with Markovian delays renamed to uncontrollable events, we couple the plant with the supervisor and export the supervised renamed discrete-event plant from Supremica. This exported file is parsed and transformed into a Markov chain using the Supremica2MRMC tool [24]. The transformation extends [12] to cater for state labels and data assignments.

We require the two additional input files, which specify the rates corresponding to the uniquely named uncontrollable delays, made during the modeling of the system, and the priority order of the controllable actions, which is employed to resolve internal nondeterminism and it is specified by the modeler. The priority of events is required as the supervised plant may contain labeled transitions and, in order to measure the performance of the plant, these labeled transitions must not introduce real nondeterministic choices [11]. In case such choices arise, the performance of the system is undefined. There are several acceptable options to resolve this issue: enrich the model with probabilistic choices that quantify the conflicting nondeterministic choices, alter the original model to eliminate them, or impose priorities on the action transitions. We chose to impose priorities on the events, obtaining directive supervision that optimizes the behavior of the supervised plant [21]. As the supervisor must not disable uncontrollable events, they are automatically given highest priority. For the rest, the priorities are specified by the modeler to be employed in Supremica2MRMC for resolution of nondeterministic choice in order to derive a pure Markov process. Ideally, the most optimal priority of events should be deduced automatically, e.g., by computing optimal schedulers as it is done for Markov decision processes [13], but this work is beyond the scope of this paper, and we schedule it for future work.

Next, we demonstrate the proposed framework in a case study involving movement coordination of automated guided vehicles of a pipeless plant.

3 Case Study: Pipeless Plant

Pipeless plants are employed in the chemical industries when there is need for great flexibility and scalability of production [19]. These plants are used to produce small amounts of high-value products, e.g., pharmaceutical or other fine chemical products, which orders or recipes change frequently over time. The main characteristic of

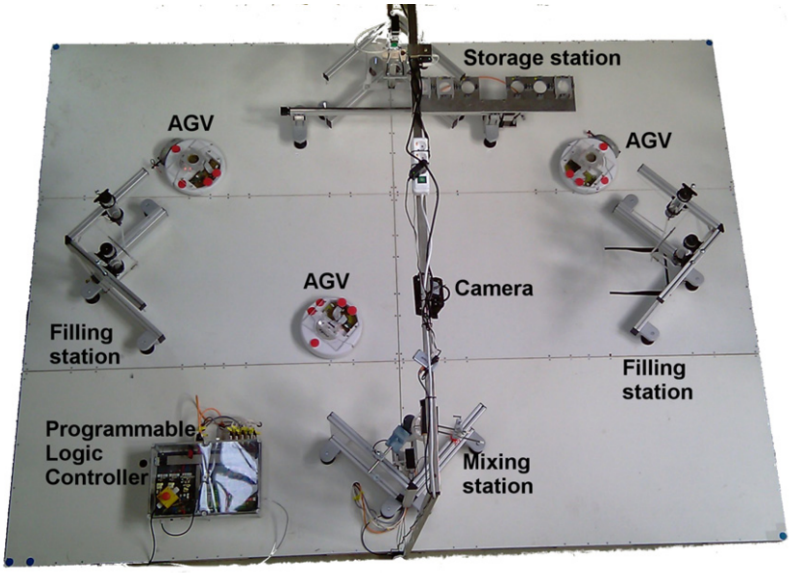

Fig. 2. Layout of the pipeless plant (adapted from [31]).

pipeless plants is that the production materials are transferred internally by means of automated guided vehicles, instead of the traditional transportation with pipes or conveyor belts. This enables greater flexibility in the production as different processing steps corresponding to different properties of the containers at different processing stations can be executed in parallel.

In the setting of this paper, we model a prototype of a pipeless (paint) plant described in [19,20]. The hardware prototype of this plant is depicted in Fig. 2 and it is currently being built by the Process Dynamics and Operations Group at the University of Dortmund, Germany. This pipeless plant comprises a storage of vessels, which are carried around using automated guided vehicles that bring the vessels to different filling and mixing stations. Filling stations can fill the vessels with colors they have at their disposal. Mixing stations blend the colors in the vessel by taking it from the vehicle and returning it when the mixing process finishes.

The control problem is to coordinate the movement of the vehicles such that all recipes, which state the colors that should be combined and mixed, are completed, without damaging the vehicles or the vessels, or pouring liquid outside of vessels. This problem has been partially addressed before by employing simulation and by modeling of hybrid properties of the movement of the vehicles [20]. In this paper, we consider high-level supervisory coordination of the movement of the vehicles, such that an arbitrary order of a set of given recipes, i.e., tasks that the vehicle should perform, is executed in order and without conflicts. We note that the vehicles have embedded control for safe acceleration/deceleration, collision avoidance, and docking [20], which makes the coordination problem relevant for supervisory control.

The basic layout of a paint manufacturing pipeless plant consists of a series of stations as depicted in Fig. 2. The stations are divided into five categories: generators, exits, filling stations, mixing stations, and waiting stations. We note

that generators, exits, and waiting stations are conceptual constructs that help us with the reasoning about the system. The generators and the exits are implemented as storage stations in the hardware, whereas a waiting station is a parking space for the vehicles. In the generators, an empty vessel is placed on the automated guided vehicle. Filling stations pour paint of particular color into the vessel that is carried by a vehicle. A filling station is capable of pouring more than one color. Mixing stations remove full vessels from vehicles, mix the paint inside them, and then place them again on a vehicle, which does not necessarily have to be the same vehicle that delivered the vessel. Finally, exit stations remove vessels whose paint has already been mixed from vehicles for storage.

We make several assumptions about the model, the most important ones being: there are no physical barriers that prevent vehicles from arriving or leaving a particular station, which is handled by the embedded control of the vehicle; vehicles are equipped with sensors and an independent control that prevents collisions with other vehicles, and they can independently reach a prescribed location in the system; only one vehicle can be accommodated at each station at a given time, with the exception of the waiting station, which can accommodate as many as necessary; once a vehicle has been instructed to go to a station, we cannot cancel the command or send it to another station before it reaches the designated station; the recipes contain predetermined order of colors that should be poured and the paint must be mixed for the recipe to be successfully completed; and the production procedure is considered as completed once the recipe sequence has been completed, there are no vessels present at the mixing stations, and all vehicles are empty and they are parked at a waiting station.

The pipeless plant that is modeled and analyzed in this case study consists of one generator, one exit, one waiting station, one mixing station, two filling stations, one of which can pour red or blue paint, and the other one can pour red or yellow paint, and two automated guided vehicles. We consider two recipes, one requiring red, blue, and yellow paint to be poured in that order, and another one requiring only red paint to be poured. For the performance evaluation metrics, we consider the sequence of recipes that first requires recipe one and, then, recipe two.

Modeling of the Plant and the Data-Based Observers

In order to obtain the plant, it is first necessary to establish what signals can be received by the controller from the system, what signals it can send to the actuators and what events are used to mark internal state changes. The signal names are given in sans serif font, whereas we specify the parameters using roman font. We overview the important signals that are communicated in the system. Indication that a particular vehicle has reached a particular station is given by the uncontrollable events $\mathsf{reach\_}\mathit{station}\mathsf{\_ag}N$, where *station* is the name of the station and N is the number of the vehicle. Indication that a particular filling station has finished pouring paint is given by the uncontrollable events $\mathsf{filled\_f}M$ where M represents the number of the filling station. Instruction for a particular filling station to pour a particular color is given by the controllable events $\mathsf{fill\_}\mathit{color}\mathsf{\_f}M$,

where M is the number of the filling station and *color* is the color that should be poured. Instruction for the mixing or exit station to grab a vessel is given by the controllable events grab_m and grab_e, respectively. Instruction for the mixing station or generator to place a vessel is given by the controllable events place_m and place_g, respectively. Instruction for a particular vehicle to go to a particular station is given by the controllable events tr_*station*_ag$N$, where *station* is the name of the destination and N is the number of the vehicle. Indication that a particular color is registered for a particular recipe is denoted by the uncontrollable events rJ_*color*, where *color* is the name of the color and J is the number of the recipe. Indication that a particular recipe is completed is given by the uncontrollable events rJ_finished for recipe J. Indication that recipe J has been assigned to vehicle N is denoted by ri_rJ_ag$N$.

We depict extended finite automata graphically, where state labels are given adjacent to the states, and event labels possibly have associated guards and variable updates below them. The uncontrollable events that represent stochastic delays have labels that begin with stoc. We note that for the sake of clarity of presentation and page limit, we do not discuss the whole model, but only several important aspects. The complete model and the tools are available from [9,24].

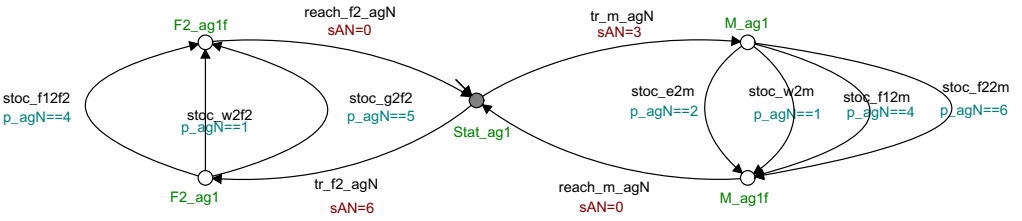

Fig. 3. Partial model of the automated guided vehicle

We depict a partial model of the automated guided vehicle in Fig. 3. The initial state is marked with an incoming arrow, whereas marked states have a gray background. The models specifies that a particular station can be reached by a vehicle, only if it has been sent to that station by the coordinator. For example, the controllable event tr_m_ag$N$ commands vehicle N to the mixing station. The vehicle responds by the uncontrollable sensing event reach_m_ag$N$ when it has reached the station. The uncontrollable events that can occur in between denote Markovian delays that represent the average time to travel to the mixing station, based on the current location of the vehicle. Since the vehicle does not carry information about its current location, the use of an observer is mandated. To identify the location of the vehicle, we employ the observer depicted in Fig. 4.

The variable p_ag$N$ codes the location of the vehicle, where value 1 denotes the waiting station, 2 denotes exit, 3 denotes the mixing station, 4 specifies the first filling station, 5 specifies the generator, and 6 specifies the second filling station. Now, by conveniently observing the location of the vehicle, we employ the assigned value of the location variable p_ag$N$ in the guard of the uncontrollable transitions that model Markovian delays in order to determines the correct delay. This is coded

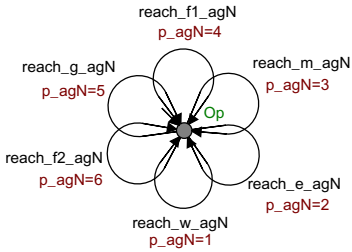

Fig. 4. Position data-based observer of the automated guided vehicle

in the uncontrollable events labeled stoc_*origin*2*destination*, which denote that amount of time need to travel from *origin* to *destination*. For example, stoc_$w2f2$ is associated with rate 1, whereas stoc_$w2m$ is associated with rate 1.15, denoting that the waiting station is closer to the mixing station than to the second filling station. We model the movement of the vehicle with respect to other stations similarly to its movement to the mixing station. In Fig. 3, we additionally show the movement to the second filling station.

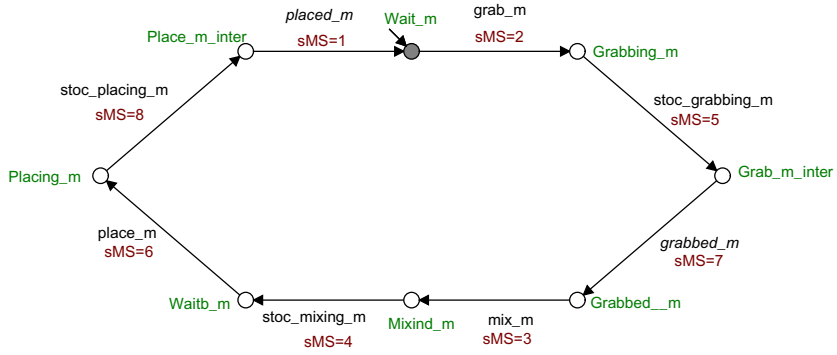

Fig. 5. Model of a mixing station

Next, we show the model of the mixing station, depicted in Fig. 5. The station is initially empty and it is ready to grab a vessel. Then it can mix its contents and when it is finished, it can place the vessel on an available vehicle and return to its initial state. The variable "sMS" tracks the current state of this automaton. The uncontrollable delays stoc_grabbing_m, stoc_mixing_m, and stoc_placing_m denote the average time of grabbing a vessel, mixing its contents, and placing it to the next available vehicle, respectively. We note that we do not require feedback when the mixing process is finished, as this is not a synchronizing event, as it is the case with grabbed_m and placed_m.

We depict the model of a filling station in Fig. 6. The station pours the corresponding only after being instructed by the coordinator. The uncontrollable event stoc_filling_*color*_fM, captures the average time needed to fill the color *color* at the filling station M. The state of the filling station is followed by the variable sFSM

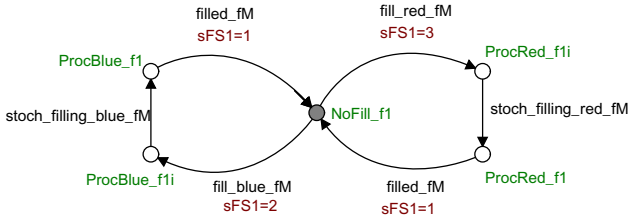

Fig. 6. Model of a filling station

for the filling station M.

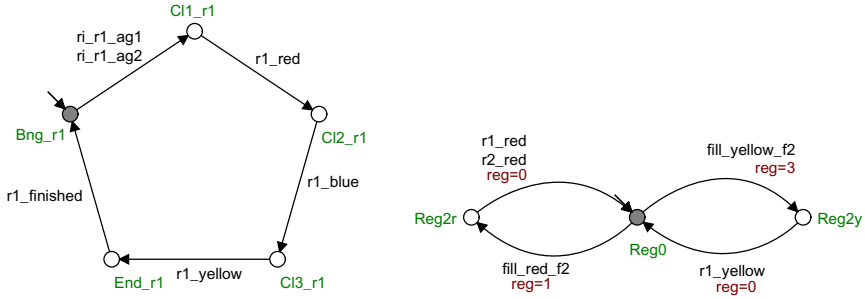

Fig. 7. a)Model of a recipe requiring the colors red, blue, and yellow to be mixed; b) Observer registering the colors that have already been filled;

The model of the recipe indicates that after the recipe has been assigned to an automated guided vehicle, events $\mathsf{ri\_r1\_ag}N$ assigning recipe 1 to vehicle N, a particular sequence of color registries has to be followed for that particular recipe, as depicted in Fig. 7a). A color registry, given by the events $\mathsf{r1\_}color$ for the color *color* for recipe 1 in Fig. 7b), is a generated event from an observer that waits for the corresponding color to be poured into the vessel from any available filling station that provided that color.

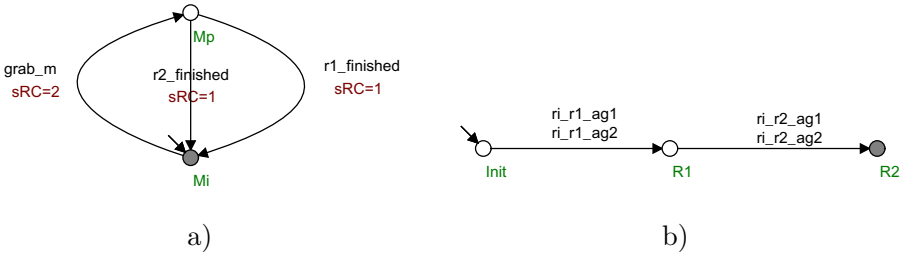

Fig. 8. a) Observer for the recipe completion; b) Recipe sequence to be carried out

We also introduce an observer for recipe completion, depicted in Fig. 8a), which indicates that recipes can only be considered complete once the mixing station grabs the corresponding vessel for mixing. Recipes are specified as sequences as shown in Fig. 8b), where the events in the sequence specify which vehicles are capable of

carrying out the specific recipe. This models the situation where different vehicles are more suitable for different recipes, e.g., some vehicles can carry heavier vessels or have specific safety features. We note that marked states ascertain that the supervisory controller completes each recipe successfully and the recipe sequence is carried out completely.

Control Requirements

We have multiple coordination constraints that we must ascertain over the plant. Most of the constraints are safety properties, whereas several can also be treated as progress properties. In total we have 22 informal movement coordination constraints, which are specified in the model [24]. Here, we discuss several characteristic coordination constraints, in order to illustrate the process of formalization of control requirements. We note that the other constraints are modeled in an analogous manner, and most of them make use of the data observers as they typically refer to locations and activities that can be performed at a specific location.

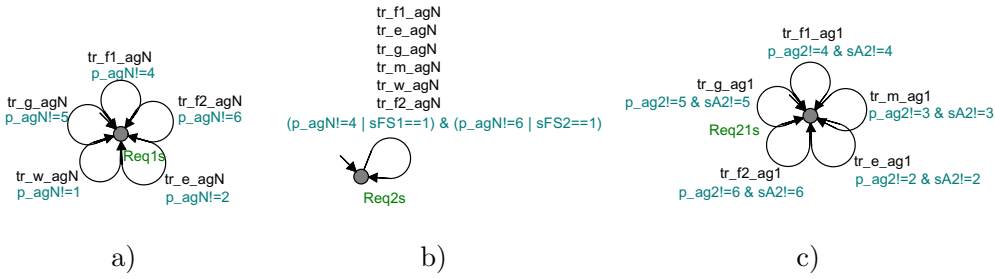

Fig. 9. Control requirements a), b), c), respectively

Coordination constraint a): "A vehicle cannot be instructed to go to a station it is already at." is an example of a progress constraint, since it does not contribute to safe behavior of the system, but only eliminates useless commands issued by the coordinator. To specify this control requirement we require the current position of the vehicle, so it is necessary to use the information registered by the position observer of Fig. 4. The requirement expresses that the events $\mathsf{tr\_}station\mathsf{\_ag}N$ are blocked at the station *station*, identified by the variable $\mathsf{p\_ag}N = station$. Fig. 9a) depicts this control requirement as specified in Supremica [1]. The control requirements are typically specified as self loops labeled by the corresponding events that are guarded by an expression that models the control requirement based on the variable assignments. Thus, for example, we can issue a command to vehicle N to go to the waiting station by employing the event $\mathsf{tr\_}w\mathsf{\_ag}N$, only if the location variable $\mathsf{p\_ag}N$ does not have value 1, which means that the vehicle is not already at the waiting station.

Coordination constraint b): "A vehicle cannot go to another station while it is being filled with paint." is a safety requirement that prevents the material from being spilt outside the vessel. This requirement is depicted in Fig 9b) and it is modeled by blocking events of the type $\mathsf{tr\_}station\mathsf{\_ag}N$, when "$\mathsf{p\_ag}N = J$", i.e.,

the vehicle is at filling station J, whereas the observer of the filling process is not in state NoFill_fJ that denotes that the filling process has terminated.

Coordination constraint c): "A vehicle cannot be commanded to move to a station, if another vehicle is already instructed or located at the same station, except for the waiting station." is a safety requirement that prevents vehicles from colliding at stations while docking. IN Fig. 9c) we show the model of this requirement for the vehicle 1 with respect to vehicle 2. The requirement is ascertained by blocking events tr_*station*_ag1 that instruct vehicle 1 to transfer to station *station*, while vehicle 2 is already at or moving toward the same station, which is detected by checking the values assigned to the variables p_ag2 and sA2.

Performance Evaluation by Stochastic Model Checking

After having modeled the plant and the control requirements, we synthesize a supervisor, as outlined in Fig. 1. Then, we process the supervised plant in which the stochastic events are renamed back from uncontrollable events, and we derive the underlying continuous-time Markov chain by employing the tool Supremica2MRMC [24]. The resulting Markov process has state labels, which correspond to the state labels of the original extended finite automata models made in Supremica. In addition, we add the data-based observations to the Markov process as state labels of the form *Variable* = *Value*. Having derived the performance model, we proceed with modeling several performance requirements and verifying them using the Markov model checker MRMC [14].

Continuous Stochastic Logic

To specify the performance requirements, we employ Continuous Stochastic Logic [3], which is completely supported by Markov model checker MRMC for continuous-time Markov chains. Here we present only a part of the logic for the needs of this paper. The logic syntax is split to state formulas and path formulas. State formulas are employed to identify states by their propositional labels or probability measures, whereas path formulas identify states that satisfy time-bounded or unbounded properties over sequences of reachable states. Both types of formulas can be coupled with probability measures to verify performance or reliability properties. State formulas SF have the following MRMC syntax [14]:

$$\texttt{SF} ::= \texttt{tt} \mid \texttt{ff} \mid \texttt{L} \mid \texttt{!SF} \mid \texttt{SF} \;\&\&\; \texttt{SF} \mid \texttt{SF} \;||\; \texttt{SF} \mid \texttt{P}\{\circ\,\texttt{p}\}[\texttt{PF}] \mid \texttt{S}\{\circ\,\texttt{p}\}[\texttt{SF}],$$

where tt denotes the constant true, ff denotes the constant false, $\texttt{L} \in \texttt{S}$ is an atomic propositional symbol, ! denotes negation, && denotes conjunction, || denotes disjunction, $\circ \in \{<, \leq, =, \geq, >\}$, and $\texttt{p} \in [0, 1]$.

The probability measure operator $\texttt{P}\{\circ\,\texttt{p}\}[\texttt{PF}]$ identifies states on paths that satisfy PF and meet the probability constraint $\circ\,\texttt{p}$. The steady-state probability measure operator $\texttt{S}\{\circ\,\texttt{p}\}[\texttt{SF}]$ checks if the steady-state probability for being in states that fulfill SF meets the probability constraint $\circ\,\texttt{p}$. These operators result in a state formulas that can be nested to make up complex path formulas [3]. Path formulas

PF can combine state formulas using the until operator, which may be bounded for a specific time interval:

$$\texttt{PF} ::= \texttt{SF U SF} \mid \texttt{SF U[r, r] SF},$$

where $r \in \mathbb{R}$ and $r \geq 0$. The until operator identifies paths that comprise states that satisfy the left state formula until they reach an end state that satisfies the right state formula in the given time interval for the bounded variant and for any time interval for the unbounded variant.

Modeling of Performance Requirements

We developed a method for domain engineers in industry that are not experienced in formulating properties in temporal logics in order to aid them with the formulation of performance requirements starting from the informal specification to expressions in CSL. We depict the method in Fig. 10. The first step is to state the outcome of the requirements explicitly. Thereafter, the relevant properties of this outcome should be identified. These properties must be expressed in terms of state or trajectories present in the system. States are identified by their state labels, or by the data observations made by the modeler.

CSL has a restricted syntax: if the performance requirement considers reachability of states, or percentage of time spent in a state, then this can be expressed. However, combinations or complex path information, are currently not supported by MRMC. For a more detailed treatment of this topic we refer to [9]. As not everything can be expressed directly, there several possibilities to be explored: the interpretation of the properties is incorrect, the original statement is not precisely specified or it is not possible to express the desired property using the available notation. In the first two cases, the performance requirement must be specified more precisely, and these are considered modeling errors. However, it happens that the notation of CSL is not sufficiently expressive for the task at hand. In some cases, it might be possible to make a compromise and weaken the original statement to obtain some useful information from the performance evaluation, but in general, it is necessary to attempt another approach. By applying this procedure, we specify several performance requirements of interest.

Performance requirement a): "What is the probability that the entire process will take less than 60 time units to complete?". For this requirement the acceptable probability bound is greater than 0.9. Thus, we have to verify whether a marked state can be reached within 60 time units with probability greater than 0.9. Such a state is identified by the following values of the corresponding observation variables $p\_ag1 = 1$, $p\_ag2 = 1$, $ld\_ag1 = 0$, and $ld\_ag2 = 0$, whereas the mixing station has to be in the state labeled $Wait\_m$. The corresponding statement is given by

```
P{ >= 0.9 }[ tt U[ 0,60 ) (p_ag1 == 1 && p_ag2 == 1 &&
                ld_ag1 == 0 && ld_ag2 == 0 && Wait_m ) ].
```

Performance requirement b): "What is the probability that the paint for the first recipe is poured before the paint for the second recipe?". For this requirements the acceptable probability bound should be greater than 0.95. Thus, we verify

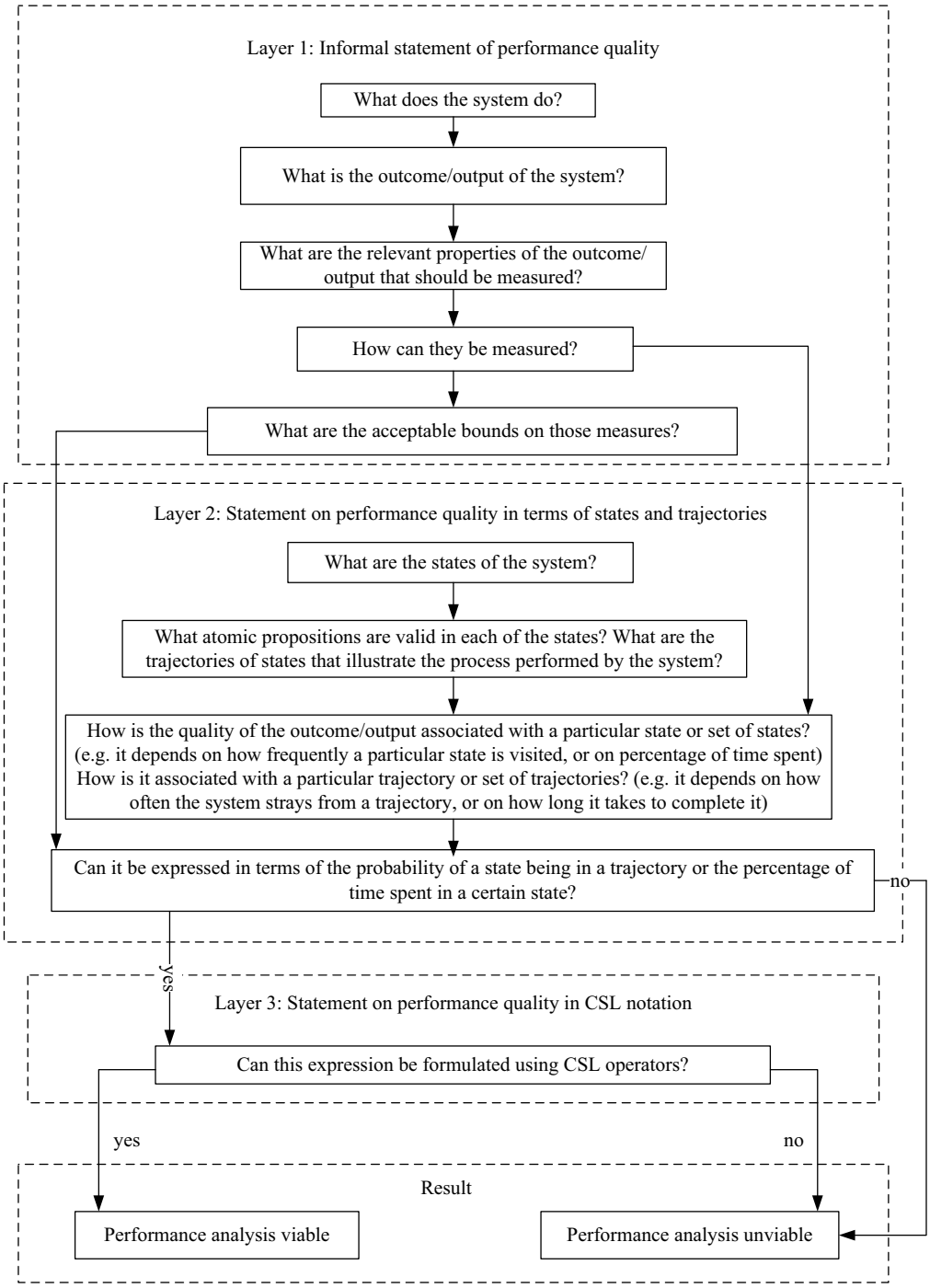

Fig. 10. Method for modeling performance requirements using CSL

whether the probability that the vehicle to which the first recipe has been assigned will reach a mixing station before a vehicle to which the second recipe has been assigned is greater than 0.95. This can be modeled by observing trajectories in which $recipe\_ag1 == 2$ and $p\_ag1 == 3$ is not valid at the same time and $recipe\_ag2 == 2$ and $p\_ag2 == 3$ are not valid at the same time, until a state is reached where $recipe\_ag1 == 1$ and $p\_ag1 == 3$ are valid or $recipe\_ag2 == 1$ and $p\_ag2 == 3$ are valid. The corresponding statement using CSL is given by

```
P{ >= 0.95 }[ (!( recipe_ag1 == 2 && p_ag1 == 3 ) &&
                !( recipe_ag2 == 2 && p_ag2 == 3 ) ) U
              ( ( recipe_ag1 == 1 && p_ag1 == 3 ) ||
                ( recipe_ag2 == 1 && p_ag2 == 3 ) ) ].
```

Performance requirement c): "What is the probability of a loaded vehicle to reside in a waiting station at least once before the process is completed?". For this requirement the acceptable bound on probability is smaller than 0.1. To express this property, we check whether the probability of residing a state where $p\_ag1 == 1$ and $ld\_ag1 == 1$ are valid or $p\_ag2 == 1$ and $ld\_ag2 == 1$ are valid is smaller than 0.1, which is given by

```
P{ <= 0.1 }[ tt U ( ( p_ag1 == 1 && ld_ag1 == 1 ) ||
                    ( p_ag2 == 1 && ld_ag2 == 1 ) ) ]
```

The verification of the properties with our set of exponential rates showed that requirement a) is met with computed probability of 0.98, whereas requirements b) and c) were not met, with probabilities 0.29 and 1, respectively.

4 Concluding Remarks

Traditionally, only safety concerns are considered during the design of supervisory controllers. To improve the design process and make it less iterative, it is important to consider the performance of the system as well. We presented an integration of supervisory control with performance evaluation that allows verification whether given performance requirements are met by the control design. To this end, the discrete-event model of the system is extended with Markovian delays. Additionally, data-based observers are introduced as a way of improving the safety requirement design, allowing for greater flexibility and simplicity in modeling. The performance was evaluated by means of Markov model checking, for which we develop a methodology for modeling performance in terms of stochastic temporal logics. The proposed framework was demonstrated on a case involving movement coordination of automated guided vehicles that transfer production materials in a pipeless plant.

References

[1] Akesson, K., M. Fabian, H. Flordal and R. Malik, *Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems*, in: *Proceedings of WODES 2006* (2006), pp. 384 – 385.

[2] Baier, C., M. Grer, M. Leucker, B. Bollig and F. Ciesinski, *Controller synthesis for probabilistic systems*, in: *Proceedings of IFIP TCS 2004* (2004), pp. 493–506.

[3] Baier, C., B. Haverkort, H. Hermanns and J.-P. Katoen, *Model-checking algorithms for continuous-time Markov chains*, IEEE Transactions on Software Engineering **29** (2003), pp. 524 – 541.

[4] Baier, C., B. R. Haverkort, H. Hermanns and J.-P. Katoen, *Model checking meets performance evaluation*, SIGMETRICS Performance Evaluation Review **32** (2005), pp. 10–15.

[5] Brázdil, T., V. Forejt and A. Kucera, *Controller synthesis and verification for Markov decision processes with qualitative branching time objectives*, in: *Automata, Languages and Programming*, Lecture Notes in Computer Science **5126**, Springer, 2010 pp. 148–159.

[6] Cassandras, C. and S. Lafortune, "Introduction to discrete event systems," Kluwer Academic Publishers, 2004.

[7] Chatterjee, K., M. Jurdzinski and T. A. Henzinger, *Simple stochastic parity games*, in: *Computer Science Logic*, Lecture Notes in Computer Science **2803**, Springer, 2003 pp. 100–113.

[8] Chen, T., T. Han and J. Lu, *On the Markovian randomized strategy of controller for Markov decision processes*, in: *Fuzzy Systems and Knowledge Discovery*, Lecture Notes in Computer Science **4223**, Springer, 2006 pp. 149–158.

[9] Estens Musa, E., "Verification of Stochastic Requirements in Supervised Plants," Master's thesis, Eindhoven University of Technology (2012).

[10] Garg, V. K., R. Kumar and S. I. Marcus, *A probabilistic language formalism for stochastic discrete-event systems*, IEEE Transactions on Automatic Control **44** (1999), pp. 280 – 293.

[11] Hermanns, H., "Interactive Markov Chains and the Quest For Quantified Quantity," Lecture Notes of Computer Science **2428**, Springer, 2002.

[12] Hermanns, H. and S. Johr, *May we reach it? or must we? in what time? with what probability?*, Proceedings of MMB 2008 (2008), pp. 1–15.

[13] Howard, R. A., "Dynamic Probabilistic Systems," John F. Wiley & Sons, 1971.

[14] Katoen, J.-P., M. Khattri and I. S. Zapreev, *A Markov reward model checker*, in: *Proceedings of QEST 2005* (2005), pp. 243 – 244.

[15] Kumar, R. and V. K. Garg, *Control of stochastic discrete event systems: Synthesis*, in: *Proceedings of CDC 1998* (1998), pp. 3299–3304.

[16] Kwong, R. H. and L. Zhu, *Performance analysis and control of stochastic discrete event systems*, in: *Feedback Control, Nonlinear Systems, and Complexity*, Lecture Notes in Control and Information Sciences **202**, Springer, 1995 pp. 114–130.

[17] Lawford, M. and W. M. Wonham, *Supervisory control of probabilistic discrete event systems*, Proceedings of Circuits and Systems 1993 **1** (1993), pp. 327 – 331.

[18] Leveson, N., *The challenge of building process-control software*, IEEE Software **7** (1990), pp. 55–62.

[19] Liefeldt, A., *Logistic simulation of pipeless plants*, in: *Logistic Optimization of Chemical Production Processes*, Wiley, 2008 pp. 37–55.

[20] Liefeldt, A. and S. Engell, *A modelling and simulation environment for pipeless plants*, in: *Proceedings of PSE 2003*, Computer-Aided Chemical Engineering **15**, Elsevier, 2003 pp. 956 – 961.

[21] Markovski, J., *Towards supervisory control of Interactive Markov chains: Controllability*, in: *Proceedings of ACSD 2011* (2011), pp. 108–117.

[22] Markovski, J., *A process-theoretic state-based framework for live supervision*, in: *Proceedings of CASE 2012* (2012), to appear. Available from [24].

[23] Markovski, J., *Process theory for supervisory control of stochastic systems with data*, in: *Proceedings of ETFA 2012* (2012), to appear. Available from [24].

[24] Markovski, J., *Tools and demo models*, `http://sites.google.com/site/jasenmarkovski` (2012).

[25] Markovski, J. and M. A. Reniers, *Verifying performance of supervised plants*, in: *Proceedings of ACSD 2012* (2012), pp. 52–61.

[26] Markovski, J., D. A. van Beek, R. J. M. Theunissen, K. G. M. Jacobs and J. E. Rooda, *A state-based framework for supervisory control synthesis and verification*, in: *Proceedings of CDC 2010* (2010), pp. 3481–3486.

[27] Miremadi, S., K. Akesson and B. Lennartson, *Extraction and representation of a supervisor using guards in extended finite automata*, in: *Proceedings of WODES 2008* (2008), pp. 193–199.

[28] Pantelic, V., S. M. Postma and M. Lawford, *Probabilistic supervisory control of probabilistic discrete event systems*, IEEE Transactions on Automatic Control **54** (2009), pp. 2013 – 2018.

[29] Ramadge, P. J. and W. M. Wonham, *Supervisory control of a class of discrete-event processes*, SIAM Journal on Control and Optimization **25** (1987), pp. 206–230.

[30] Schiffelers, R. R. H., R. J. M. Theunissen, D. A. v. Beek and J. E. Rooda, *Model-based engineering of supervisory controllers using CIF*, Electronic Communications of the EASST **21** (2009), pp. 1–10.

[31] Schoppmeyer, C., M. Hüfner, S. Subbiah and S. Engell, *Timed automata based scheduling for a miniature pipeless plant with mobile robots*, in: J.-F. Zhang and D. Sauter, editors, *Proc. 2012 IEEE Multiconference on System and Control* (2012).

[32] Skoldstam, M., K. Akesson and M. Fabian, *Modeling of discrete event systems using finite automata with variables*, in: *Proceedings of CDC 2007* (2007), pp. 3387–3392.