Modeling guidelines for component-based supervisory control synthesis*

Martijn Goorden[1], Joanna van de Mortel-Fronczak[1], Michel Reniers[1],
Wan Fokkink[1,2], and Jacobus Rooda[1]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands,
{m.a.goorden, j.m.v.d.mortel, m.a.reniers, j.e.rooda}@tue.nl
[2] Vrije Universiteit, Amsterdam, The Netherlands, w.j.fokkink@vu.nl

Abstract. Supervisory control theory provides means to synthesize supervisors from a model of the uncontrolled plant and a model of the control requirements. Currently, control engineers lack experience with using automata for this purpose, which results in low adaptation of supervisory control theory in practice. This paper presents three modeling guidelines based on experience of modeling and synthesizing supervisors of large-scale infrastructural systems. Both guidelines see the model of the plant as a collection of component models. The first guideline expresses that independent components should be modeled as asynchronous models. The second guideline expresses that physical relationships between component models can be easily expressed with extended finite automata. The third guideline expresses that the input-output perspective of the control hardware should be used as the abstraction level. The importance of the guidelines is demonstrated with examples from industrial cases.

Keywords: Supervisory control synthesis · Automata · Modeling.

1 Introduction

The design of supervisors for cyber-physical systems has become a challenge, as these systems include more and more components to control and functions to fulfill, while at the same time market demands require verified safety, decreasing costs, and decreasing time-to-market. Model-based systems engineering methods can help in overcoming these difficulties, see [25].

For the design of supervisors, the supervisory control theory of Ramadge-Wonham [23, 24] provides means to synthesize supervisors from a model of the uncontrolled plant (describing what the system *can* do) and a model of the control requirements (describing what the system *may* do). Such a supervisor interacts with the plant by dynamically disabling some controllable events. Then synthesis guarantees by construction that the closed-loop behavior of the supervisor and the plant adheres to all requirements and, furthermore, is nonblocking, controllable, and maximally permissive.

* Supported by Rijkswaterstaat, part of the Dutch Ministry of Infrastructure and Water Management.

The number of industrial applications of supervisory control theory reported in literature is low. In [38], two reasons are provided for this. First, it refers to the lack of tooling with sufficient computational strength to cope with the size of industrial applications. Second, it mentions the "lack of experience among control engineers with modeling and specification in the framework of automata".

Papers that do publish industrial cases often present only the final model and not the journey to arrive at this model. This makes it hard to disseminate knowledge about modeling a system for the purpose of supervisory control synthesis towards practitioners. A few exceptions exist in literature. The authors of [7,14] have indicated that modeling the system and its requirements is difficult and introduced concepts like, e.g., templates to assist the engineer in modeling correctly, i.e., such that the obtained models exhibit the behavior the engineer intended to model. The description of the case study in [4] is annotated with modeling choices, yet they are not generalized into a modeling method. In [29], a method for modeling cyber-physical systems is presented utilizing template-based modeling of [7,14]. Finally, several modeling guidelines are proposed in [35] based on experience with modeling bagage-handling systems, which is described and modeled in [34].

The purpose of this paper is to provide three modeling guidelines based on experience of modeling and synthesizing supervisors of large-scale infrastructural systems [19,28,29]. The first modeling guideline expresses that independent plant components should be modeled as asynchronous plant models, i.e., having no shared events. The second modeling guideline recommends that physical relationships between component models can be easily expressed with extended finite automata. The third modeling guideline expresses to use the abstraction level of the inputs and outputs of the control hardware for the plant models. These three modeling guidelines extend the set of modeling guidelines previously published in [10,11].

The effect of the first modeling guideline is that each individual plant model is modeled as small as possible. Besides that smaller models are easier to understand and maintain over time, having smaller plant models has a significant positive effect on the efficiency of module-based synthesis techniques, like modular synthesis of [22] and multilevel synthesis of [15], as shown in [11]. The second guideline is a natural extension to the first guideline. It may be that two independent components become dependent by their arrangement in the system, i.e., there is a physical relationship that relates the behavior of these two components together. The result of the second guideline is a set of asynchronous component models for the components and an additional automata modeling the physical relationship. The third guideline helps in determining the right abstraction level of the model. The result of following the guideline is that the first and second guideline are more often applicable.

Requirement specifications in practice often violate the aforementioned guidelines. Although the guidelines may sound somewhat obvious, it required several real-life case studies with supervisory control synthesis, see [19,28,29], to formulate them and grasp their importance.

The paper is structured as follows. Section 2 provides the preliminaries of this paper. Section 3 continues by discussing the guideline concerning the modeling of independent plant components. Section 4 discusses how to model the physical dependencies between otherwise independent plant components. In Section 5, the guideline concerning the input-output perspective is discussed. The paper concludes with Section 6.

2 Preliminaries

This section provides a brief summary of concepts related to automata and supervisory control theory relevant for this paper. These concepts are taken from [2,38]. We first explain supervisory control synthesis for automata conceptually. Sections 2.1-2.3 introduce these concepts formally.

The supervisory control theory of Ramadge-Wonham [23,24] provides means to synthesize supervisors from an automaton model of the uncontrolled plant and an automaton model of the control requirements. For industrial-size systems, the plant model and requirement model are each composed of smaller models describing a component of a system or a part of the desired behavior, respectively, where the smaller models synchronize by shared events. When a system controlled by a supervisor adheres to all specified requirements, the supervisor is called *safe*.

A supervisor interacts with the plant by dynamically disabling events. For the purpose of supervisory control synthesis, all events are classified either as controllable or as uncontrollable. Controllable events may be disabled by the supervisor, such as turning an actuator on; uncontrollable events may not be disabled by the supervisor, such as a sensor switching value. A supervisor adhering to this notion is called *controllable*.

The automata of the plant and requirement models also have marked states, which represent a final state or a safe mode-of-operation. It is desired that a controlled system should always be able to reach at least one of the marked states. A supervisor ensuring this is called *nonblocking*.

Finally, a trivial, yet undesired, supervisor is often one that disables all controllable events in order to be safe, controllable, and nonblocking. Therefore, a more desired supervisor is one that restricts the system only when it is needed to enforce safety, controllability, and nonblockingness. Such a supervisor is called *maximally permissive*. Supervisory control synthesis guarantees by construction that the supervisor is safe, nonblocking, controllable, and maximally permissive.

2.1 Finite automata

An automaton is a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the (finite) state set, Σ is the (finite) set of events also called the alphabet, $\delta : Q \times \Sigma \to Q$ the partial function called the transition function, $q_0 \in Q$ the initial state, and $Q_m \subseteq Q$ the set of marked states. The alphabet $\Sigma = \Sigma_c \cup \Sigma_u$ is partitioned into sets containing the controllable events (Σ_c) and the uncontrollable events (Σ_u), and Σ^* is the set of all finite strings of events in Σ, including empty string ε.

We denote with $\delta(q, \sigma)!$ that there exists a transition from state $q \in Q$ labeled with event σ, i.e., $\delta(q, \sigma)$ is defined. The transition function can be extended in the natural way to strings as $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$ where $s \in \Sigma^*$, $\sigma \in \Sigma$, and $\delta(q, s\sigma)!$ if $\delta(q, s)! \wedge \delta(\delta(q, s), \sigma)!$. We define $\delta(q, \varepsilon) = q$ for the empty strings. The language generated by the automaton G is $\mathcal{L}(G) = \{s \in \Sigma^* \mid \delta(q_0, s)!\}$ and the language marked by the automaton is $\mathcal{L}_m(G) = \{s \in \Sigma^* \mid \delta(q_0, s) \in Q_m\}$.

A state q of an automaton is called reachable if there is a string $s \in \Sigma^*$ with $\delta(q_0, s)!$ and $\delta(q_0, s) = q$. A state q is coreachable if there is a string $s \in \Sigma^*$ with $\delta(q, s)!$ and $\delta(q, s) \in Q_m$. An automaton is called nonblocking if every reachable state is coreachable.

Two automata can be combined by synchronous composition. In a synchronous composition, transitions labeled with shared events have to be executed simultaneously.

Definition 1. *Let $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$, $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ be two automata. The synchronous composition of G_1 and G_2 is defined as*

$$G_1 \parallel G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1\|2}, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2})$$

where

$$\delta_{1\|2}((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \delta_1(x_1, \sigma)!, \\ & \text{and } \delta_2(x_2, \sigma)! \\ (\delta_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } \delta_1(x_1, \sigma)! \\ (x_1, \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } \delta_2(x_2, \sigma)! \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Synchronous composition is associative and commutative up to reordering of the state components in the composed state set. Two automata are called asynchronous if no events are shared, i.e., they do not synchronize over any event.

A composed system $\mathcal{G}$ is a collection of automata, i.e., $\mathcal{G} = \{G_1, \ldots, G_m\}$. The synchronous composition of a composed system $\mathcal{G}$, denoted by $\parallel \mathcal{G}$, is defined as $\parallel \mathcal{G} = G_1 \parallel \ldots \parallel G_m$, and the synchronous composition of two composed systems $\mathcal{G}_1 \parallel \mathcal{G}_2$ is defined as $(\parallel \mathcal{G}_1) \parallel (\parallel \mathcal{G}_2)$. A composed system $\mathcal{G} = \{G_1, \ldots, G_m\}$ is called a product system if the alphabets of the automata are pairwise disjoint, i.e., $\Sigma_i \cap \Sigma_j = \emptyset$ for all $i, j \in [1, m], i \neq j$ [24].

Finally, let G and K be two automata with the same alphabet Σ. K is said to be controllable with respect to G if, for every string $s \in \Sigma^*$ and $u \in \Sigma_u$ such that $\delta_K(q_{0,K}, s)!$ and $\delta_G(q_{0,G}, su)!$, it holds that $\delta_K(q_{0,K}, su)!$, where the subscript G refers to elements of G and subscript K refers to elements of K.

2.2 Extended finite automata

In [32], extended finite automata (EFAs) are introduced for modeling systems, which are FAs augmented with bounded discrete variables. An EFA is a 7-tuple $E = (L, V, \Sigma, \rightarrow, l_0, v_0, L_m)$, where L is the (finite) location set, V the set of variables, Σ is the (finite) set of events also called the alphabet, $\rightarrow$ the extended transition relation, $l_0 \in L$ the initial location, v_0 the initial valuation, and $L_m \subseteq L$ the set of marked locations.

In an EFA, the transition relation is enhanced with guard expressions (conditions) and variable assignments (updates). Formally, the extended transition relation is $\rightarrow$: $L \times C \times \Sigma \times U \times L$, where C is the set of all conditions and U the set of all updates. A transition is enabled if the associated condition evaluates to true for the current variables valuation. After taking a transition, the variables valuation is updated according to the associated update.

A condition is a Boolean expression constructed from discrete variables, location variables, constants, the Boolean literals true (**T**) and false (**F**), and the usual arithmetical operators and logical connectives, see [20]. A location variable is a reference to a location, denoted by A.l, where A is the automaton name and l a location of automaton A. It evaluates to **T** when A is in location l.

An update consists of zero or more variable assignments of the form $v_b := c$, where ':=' denotes an assignment of the value of c to variable v_b. It is not allowed for an update to have multiple assignments for the same variable.

Two EFAs can be combined by computing the synchronous product as defined in [32]. The state of an EFA is the combination of the active location and current variables valuation. With respect of FAs, two EFAs are now called asynchronous if they do not share events, variables, or location variables.

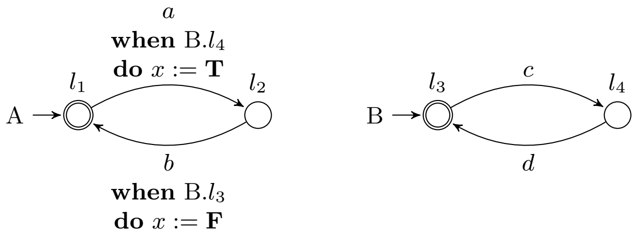


Fig. 1. An example of two EFAs.

Figure 1 shows an example of two EFAs. As shown with EFA A, keyword **when** indicates the condition of the transition and keyword **do** indicates the update. In EFA B the condition and update are omitted. An omitted condition indicates that the condition for that transition is **T**. An omitted update indicates an 'I don't care' update, i.e., the value of the variables is updated by another synchronizing transition or, when no synchronizing transitions update the variable, the value remains the same.

State-based expressions are introduced in [16, 17] as a modeling formalism more closely related to the textual formulation of control requirements. The state-event expression e **needs** c formulates that event e is only enabled when condition c evaluates to **T**. The EFA representation of a state-event expression is shown in Figure 2, such that the synchronous product of two EFAs can be used to synchronize state-based expressions with EFAs or other state-based expressions.

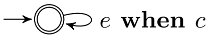


Fig. 2. The EFA representation of state-event expression e **needs** c.

2.3 Supervisory control theory

The objective of supervisory control theory is to design an automaton called a supervisor which function is to dynamically disable controllable events so that the closed-loop system of the plant and the supervisor obeys some specified behavior, see [2,23,24,38]. More formally, given a plant model P and requirement model R, the goal is to synthesize supervisor S that adheres to the following control objectives.

- *Safety*: all possible behavior of the closed-loop system $P \parallel S$ should always satisfy the imposed requirements, i.e., $\mathcal{L}(P \parallel S) \subseteq \mathcal{L}(P \parallel R)$
- *Controllability*: uncontrollable events may never be disabled by the supervisor, i.e., S is controllable with respect to P.
- *Nonblockingness*: the closed-loop system should be able to reach a marked state from every reachable state, i.e., $P \parallel S$ is nonblocking.
- *Maximal permissiveness*: the supervisor does not restrict more behavior than strictly necessary to enforce safety, controllability, and nonblockingness, i.e., for all other supervisors S' satisfying safety, controllability, and nonblockingness it holds that $\mathcal{L}(P \parallel S') \subseteq \mathcal{L}(P \parallel S)$.

Given a composed system representation of the plant $\mathcal{P} = \{P_1, \ldots, P_m\}$ and a collection of requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$, we define the tuple $(\mathcal{P}, \mathcal{R})$ as the *control problem* for which we want to synthesize a supervisor. Furthermore, in the context of supervisory control synthesis we call each model $P_i \in \mathcal{P}$ a *component* model, to differentiate it from the plant model $P = \parallel \mathcal{P}$.

In this paper, three different synthesis techniques are discussed: monolithic synthesis, modular synthesis, and multilevel synthesis. These synthesis techniques are introduced below.

Monolithic supervisory control synthesis results in a single supervisor S from a single plant model and a single requirement model, see [23] or, in case of EFAs, see [20]. There may exist multiple automata representations of the maximally permissive, safe, controllable, and nonblocking supervisor. When the plant model and the requirement model are given as a composed system $\mathcal{P}$ and $\mathcal{S}$, respectively, the monolithic plant model P and requirement model R are obtained by performing the synchronous composition of the models in the respective composed system.

Modular supervisory control synthesis uses the fact that the desired behavior is often specified with a collection of requirements $\mathcal{R}$ [37]. Instead of first transforming the collection of requirements into a single requirement, as monolithic synthesis does, modular synthesis calculates for each requirement a supervisor based on the plant model. In other words, given a control problem $(\mathcal{P}, \mathcal{R})$ with $\mathcal{R} = \{R_1, \ldots, R_n\}$, modular synthesis solves n control problems $(\mathcal{P}, \{R_1\}), \ldots, (\mathcal{P}, \{R_n\})$. Each control problem $(\mathcal{P}, \{R_i\})$ for $i \in [1, n]$ results in a safe, controllable, nonblocking, and maximally permissive supervisor S_i. Unfortunately, the collection of supervisors $\mathcal{S} = \{S_1, \ldots, S_n\}$ can be conflicting, i.e., $P \parallel S_1 \parallel \ldots \parallel S_n$ can be blocking. A nonconflicting check can verify whether $\mathcal{S}$ is nonconflicting, see [6, 18, 21]. In the case that $\mathcal{S}$ is nonconflicting, $\mathcal{S}$ is also safe, controllable, nonblocking, and maximally permissive for the original control problem $(\mathcal{P}, \mathcal{R})$ [37]. In the case that $\mathcal{S}$ is conflicting, an additional coordinator C can be synthesized such that $\mathcal{S} \cup \{C\}$ is safe, controllable, nonblocking, and maximally permissive for the original control problem $(\mathcal{P}, \mathcal{R})$, see [33].

An extension to this approach, as proposed by [22], states that instead of synthesizing each time with the complete plant $\mathcal{P}$, it suffices to only consider those automata that relate to the requirement that is considered. This extension is used in the remainder of this paper.

Multilevel supervisory control synthesis is inspired by decompositions of systems by engineers [15]. For each subsystem, a supervisor is synthesized based on requirements for only those subsystems. For synthesis, this resembles modular supervisory control in the sense that for multilevel synthesis requirements related to the same subsystem are grouped together before synthesis is performed, and a supervisor is synthesized for each such subsystem. Again, the collection of synthesized supervisors may be conflicting.

Requirements relate different component models, as events and variables mentioned in a requirement should originate from the component models. Multilevel synthesis allows to apply synthesis to a subsystem of component and requirement models, as long as all component models related to these requirement models are included in this subsystem. Therefore, it is important to formulate small requirement models, as shown in [11].

3 Modeling independent components

The first modeling guideline concerns the modeling of the plant. Industrial systems consist of numerous components or subsystems, of which many are clearly acting asynchronously in the uncontrolled situation. Consider for example two conveyor belts after each other, each actuated by its own motor. In the uncontrolled situation, these actuators can behave independently of each other. For the plant model, these two actuators are modeled by two asynchronous automata. Therefore, the first guideline is formulated as follows.

Model independent plant components as asynchronous component models.

Plant components that have no relationship with each other, should not be combined into a single component model. A single component model suggests a relationship, which is absent in this case. Having asynchronous models (i.e., no shared events, variables, or location variables) increases readability of the model, but also allows divide-and-conquer strategies to synthesize supervisors for smaller subsystems. We illustrate this with two examples.

3.1 Autonomous robot

In this section, the modeling guideline will be illustrated with an industrial example. Consider an autonomous omnidirectional robot that can move on a factory floor along a grid, described by the application published in [8]. The goal of the supervisor is to ensure safe operation of the robot on the factury floor. We want to model the pose of the robot, i.e., the combination of position along the x-axis, position along the y-axis, and orientation of the front of the robot.

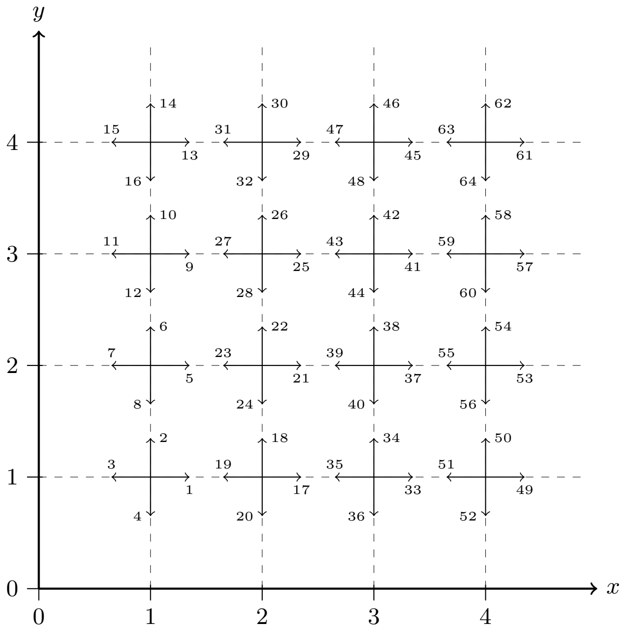


Fig. 3. The schematic representation of the factory floor. The numbers represent states, with a state being the combination of x-position, y-position, and orientation.

Figure 3 shows the schematic representation of the factory floor, where the x-axis, y-axis, and orientation are each discretized into four possible values. Each arrow indicates a pose of the robot: an x-position, y-position, and orientation. These poses are the states of the robot. The number indicates the state number that will be used in modeling this environment. Note that in this example the initial state is not explicitly modeled. Any state could act as the initial state.

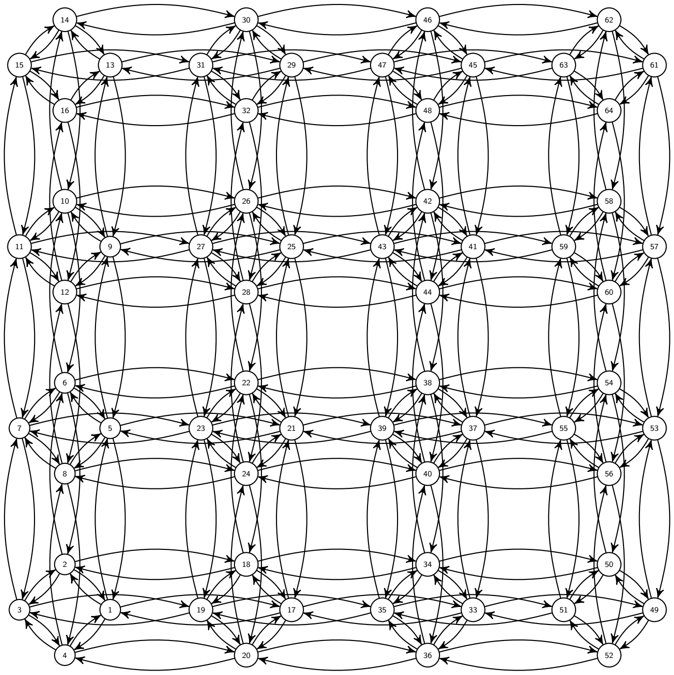


Fig. 4. The factory floor modeled as a single plant model P. Event labels are not depicted.

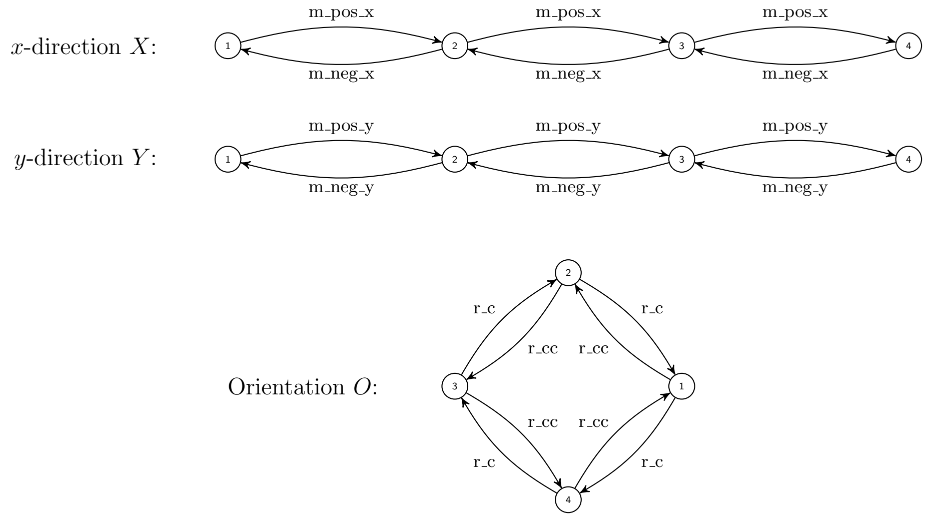


Fig. 5. The factory floor modeled as three asynchronous component models.

Figure 4 shows how this factory floor could be modeled as a single component model P along the lines of [8]. This dense plant model is hard to read. The way this model is depicted unveils that there is structure in this system, which could be exploited further. Figure 5 shows three asynchronous component models X, Y, and O that in a synchronous composition model exactly the same behavior, i.e., $P = X \parallel Y \parallel O$.

For modular and multilevel synthesis, having multiple asynchronous component models instead of a single large model is an advantage. This can be demonstrated with the following requirements. Suppose that the autonomous omnidirectional robot may only move in a certain direction if it is oriented in that direction. This requirement can be formalized as follows. Requirements $R_1, \ldots, R_4$ use the single component model P, while $R'_1, \ldots, R'_4$ use the asynchronous component models X, Y, and O. Understanding and assessing the correctness of requirements $R_1, \ldots, R_4$ is more difficult than that of requirements $R'_1, \ldots, R'_4$.

$$\begin{aligned}
R_1 : \quad & \text{m\_pos\_x } \textbf{needs } P.1 \vee P.5 \vee P.9 \vee P.13 \vee P.17 \vee P.21 \vee P.25 \vee P.29 \\
& \qquad \vee P.33 \vee P.37 \vee P.41 \vee P.45 \\
R_2 : \quad & \text{m\_neg\_x } \textbf{needs } P.19 \vee P.23 \vee P.27 \vee P.31 \vee P.35 \vee P.39 \vee P.43 \vee P.47 \\
& \qquad \vee P.51 \vee P.55 \vee P.59 \vee P.63 \\
R_3 : \quad & \text{m\_pos\_y } \textbf{needs } P.2 \vee P.6 \vee P.10 \vee P.18 \vee P.22 \vee P.26 \vee P.34 \vee P.38 \\
& \qquad \vee P.42 \vee P.50 \vee P.54 \vee P.58 \\
R_4 : \quad & \text{m\_neg\_y } \textbf{needs } P.8 \vee P.12 \vee P.16 \vee P.24 \vee P.28 \vee P.32 \vee P.40 \vee P.44 \\
& \qquad \vee P.48 \vee P.56 \vee P.60 \vee P.64
\end{aligned}$$

$$\begin{aligned}
R'_1 : \quad & \text{m\_pos\_x } \textbf{needs } O.1 \\
R'_2 : \quad & \text{m\_neg\_x } \textbf{needs } O.3 \\
R'_3 : \quad & \text{m\_pos\_y } \textbf{needs } O.2 \\
R'_4 : \quad & \text{m\_neg\_y } \textbf{needs } O.4
\end{aligned}$$

In case of the single component model P, we obtain the four control problems $(P, R_i), i \in [1, 4]$. In case of the asynchronous component models X, Y, and O, we obtain the four control problems $(X \parallel O, R'_1)$, $(X \parallel O, R'_2)$, $(Y \parallel O, R'_3)$, and $(Y \parallel O, R'_3)$. Table 1 shows numerical results for synthesizing modular supervisors for the control problems mentioned before. For each supervisor, the number of states and transitions is mentioned. By modeling the subsystem as a set of asynchronous automata models, a reduction in the size of the supervisors is obtained. This reduction can be even more significant if a finer discretization is used. Assume that both the x and y directions are discretized in k values. Each of the four synthesized supervisors using the single component model has $4k^2$ states and $21k^2 - 13k$ transitions; each of the four synthesized supervisors

Table 1. Experimental results for synthesizing modular supervisors with the single component model and the multiple components model, with the monolithic supervisor as reference. The states and transitions are of the state space of each supervisor and $i \in \{1, 2, 3, 4\}$.

Model	Supervisor	States	Transitions
Single plant	S_i	64	284
Multiple plants	S'_i	16	47
Monolithic supervisor	S	64	176

using asynchronous component models has $4k$ states and $13k - 4$ transitions. So, instead of the supervisors growing quadratic in k, the overall state space can be reduced to only growing linearly in k.

3.2 Waterway lock

In this section, the effect of the modeling guideline on the efficiency of module-based synthesis techniques is demonstrated with a large-scale industrial example. Consider a waterway lock in a river or a canal, which is an infrastructural system that maintains a difference in water levels at both sides while also allowing ships to go from one water level to the other water level. Such a system consists of actuators, such as motors to open gates, sensors, such as measuring whether a gate is open, traffic lights, to communicate with vessels, and buttons, for an operator to interact with the system.

A model of Lock III, located in Tilburg, the Netherlands, is presented in [28]. This model adheres to the proposed modeling guideline of using asynchronous component models for independent plant components. This model is adjusted such that it ignores the modeling guideline. For example, all independent components of the gate actuators on the upstream side of the lock are combined.

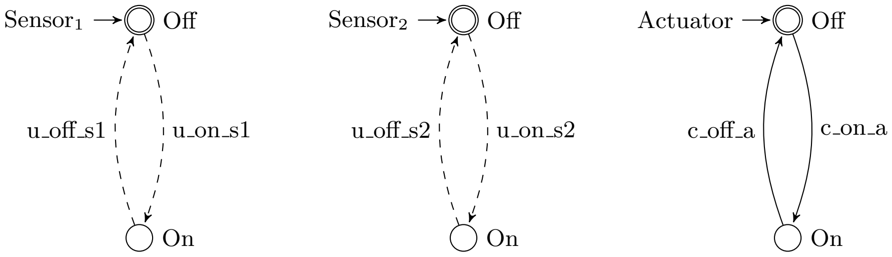
Table 2. Experimental results for synthesizing modular and multilevel supervisors with the models of Lock III violating or adhering to the modeling guideline.

Model	Components	Supervisor	States
Violating the guideline	35	Monolithic	$6.0 \cdot 10^{24}$
		Multilevel	$1.4 \cdot 10^{21}$
		Modular	$1.8 \cdot 10^{9}$
Adhering to the guideline	51	Monolithic	$6.0 \cdot 10^{24}$
		Multilevel	$3.5 \cdot 10^{7}$
		Modular	$6.8 \cdot 10^{5}$

Table 2 shows experimental results of synthesizing supervisors for the two different versions of the model of Lock III. By adhering to the modeling guideline, the number of component models increases from 35 automata to 51 automata. This has a significant effect on the efficiency of multilevel and modular synthesis. For the model violating the guideline, the combined size of the supervisors, which is the sum of the size of each individual supervisor, for multilevel synthesis is $1.4 \cdot 10^{21}$ states, which can be significantly reduced to $3.5 \cdot 10^{7}$ states if one adheres to the modeling guideline. If one adheres to the modeling guideline and deploys modular synthesis, the combined size of the supervisors can be even reduced to $6.8 \cdot 10^{5}$ states. These results clearly indicate the relevance of the proposed modeling guideline in practice.

4 Modeling physical relations

The second modeling guideline concerns the modeling of physical relations between components or subsystems. For cyber-physical systems, most actuators and sensors behave independent of each other, see the first modeling guideline in Section 3. Yet, some actuators and sensors are related with each other through the physical design of the component or subsystem. For example, consider a hydraulic arm, which can extend and retract, and two sensors measuring the end position, one for the fully extended position of the arm and one for the fully retracted position. If no faults occur, then these two sensors are never activated at the same time, as it is physically impossible that the hydraulic arm is fully extended and fully retracted at the same time.



$R:\quad$ c_off_a **needs** $\text{Sensor}_1.\text{On} \wedge \text{Sensor}_2.\text{On}$

Fig. 6. Models to illustrate the issue with omitting physical relations. Solid arrows indicate transitions labeled with controllable events, dashed transitions indicate transitions labeled with uncontrollable events.

In [39] the importance of modeling physical relationships is shown. The authors argue that models that are nonblocking, like a synthesized supervisor, may become blocking when they are implemented on actual control hardware. This is illustrated with the models shown in Figure 6. In this example, two sensors and an actuator are modeled without any physical relationship between them (so the modeling guideline from Section 3 has been applied). Requirement R expresses that the actuator may only be turned off when both sensors are on. A safe, nonblocking, controllable, and maximally permissive supervisor synthesized from these plant and requirement models disables event c_off_a when both sensors are not on at the same time and enables always all other events. When this supervisor is implemented on the system where a physical relation ensures that both sensors can never be on at the same time, the controlled system is no longer nonblocking. As event c_on_ is always enabled, the actuator can reach the state On. Subsequently, event c_off_a is permanently disabled by the physical relation between the sensors, so the actuator cannot reach a marked location from the reachable location On.

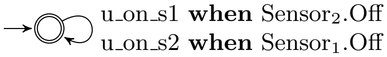

u_on_s1 **when** Sensor$_2$.Off
u_on_s2 **when** Sensor$_1$.Off

Fig. 7. The EFA model representing the physical relationship between the two sensors in the example of Figure 6. In this drawing, the two transitions, each labeled with a different event, are visualized with only a single edge as they have the same source and target state.

EFAs are very suitable to include physical relationships into the plant model. By deploying EFAs, the actuators and sensors can be first modeled as if they do not have any physical relationship, resulting in asynchronous component models. Subsequently, a component model can be added explicitly, modeling the physical relationship. Figure 7 models the physical relationship between the two sensor models from the example in Figure 6. This model shows clearly that Sensor$_1$ can only go on when Sensor$_2$ is off and vice versa. This example demonstrates that using EFAs for modeling physical relationships provides a clear and well maintainable model. Therefore, the second guideline is formulated as follows.

Model physical relationships between components with EFAs.

The proposed method of first modeling sensors and actuators with asynchronous component models and subsequently modeling the physical relationship with EFAs maintains the component-based modeling approach. Three other modeling approaches used in literature do not adhere to the proposed modeling guideline. The first method is to model the physically related components directly as a single component model, see, for example, the several sensors in the model of Lock III [28]. The second method is to model the physical relationship with an additional FA model, which is essentially the first method yet now keeping the original component models, see, for example, the interaction between actuators and sensors in the model of an MRI scanner [36]. The third method is to model the physical relationship directly in one of the related components, see, for example, the relationship between sensors and actuators in the model of Lock III [28].

The proposed modeling guideline has no impact on the efficiency of module-based synthesis algorithms like modular and multilevel synthesis. When synthesis is performed for a particular (set of) requirement(s), not only the directly related component models are selected, but also those indirectly related. Therefore, the models obtained by following the guideline or the other three methods mentioned in the paragraph above all have the same state-space representation in their synchronous product of the component models. The advantage of using the proposed modeling guideline is primarily in ease of modeling, understanding the model, and adjusting the model.

5 Modeling with the input-output perspective

The third modeling guideline concerns the abstraction level of the model. Choosing the 'right' abstraction level for the model is often not straightforward. Often systems are modeled with high-level events, such as starting a machine, handing over a product to a buffer, or moving a robot to a certain location. In [1], the implementation of the supervisor on control hardware is considered, leading to a so-called input-output perspective modeling approach. With this perspective, events relate to the change of signal value sent to actuators or received from sensors. Furthermore, all events related to actuators are controllable and all events related to sensors are uncontrollable. It turns out that this input-output perspective has several advantages, which is explained next. Therefore, the third guideline is formulated as follows.

Use the abstraction level of the inputs and outputs of the control hardware for the plant model.

For the case studies with infrastructural systems, the goal was to eventually deploy the synthesized supervisor on hardware. Choosing the abstraction level of the inputs and outputs of the control hardware allows for the generation of control code, see [3].

Furthermore, this abstraction level leads to many small and loosely coupled models of the sensors and actuators, based on just a few templates, as introduced in [14]. Supervisory control synthesis benefits from having (almost) a product system, see Section 3 and the work of [5, 9, 12, 26, 35]. In software engineering, this modeling method is called component-based modeling, see [13].

Using the input-output perspective for modeling can ultimately result in skipping synthesis completely, as shown in [9]. By using the input-output perspective, textual control requirements formulated by engineers can be more easily translated into models, as the states of actuators and sensors are directly available in the plant model. This turns out to be beneficial for supervisory control synthesis, as the plant models and requirement models together already form a safe, controllable, nonblocking, and maximally permissive supervisor and no synthesis is needed.

Another modeling method called product-based modeling should be avoided when possible. An example of a model with this abstraction level is the wafer scanner logistics model of [31]. It was not possible to synthesize a monolithic supervisor for this model. In the PhD thesis [30], the wafer scanner is modeled on the action level without products (towards the input-output perspective, yet not fully there). For this adapted model, a monolithic supervisor has been synthesized. In Section 5.1 this example will be discussed in more detail.

5.1 Industrial examples

In this section, modeling with the input-output perspective is demonstrated. For this purpose, three different case studies are discussed.

Production line buffer

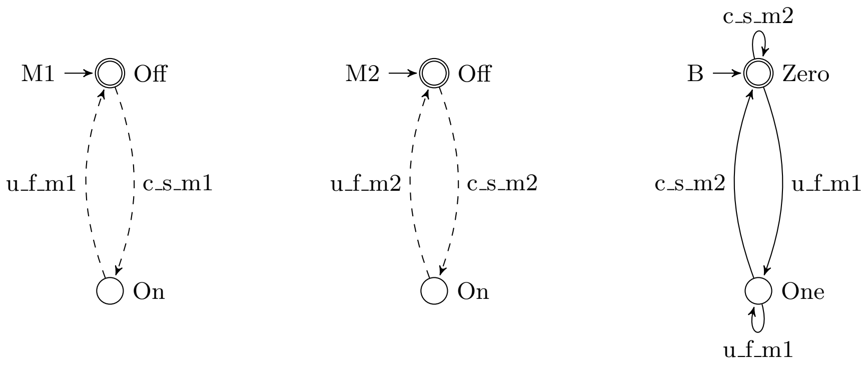


Fig. 8. A model of two machines M1 and M2 and a connecting buffer B. The letter c is an abbreviation for controllable, u for uncontrollable, s for start, and f for finish.

In the first example in Figure 8, we model two machines M1 and M2 and a buffer B connecting the two machines, adapted from [38]. The two machines display similar behavior. The two states indicate that the machine is either on or off. A controllable event can start the machine, and an uncontrollable event indicates that the machine has finished. The start and finish events are also used in the model of the buffer. A product is placed in the buffer when the first machine is finished, and a product is taken from the buffer when the second machine starts. As can be seen, a high abstraction level is chosen to model the system.

The FESTO production line, as described and modeled in [27], also contains a buffer to temporarily store products between two work stations. This example shows which actuators and sensors are connected to the inputs and outputs of the control hardware. There is an actuator A1 present to move a product from the previous work station to the buffer, an actuator A2 to move a product from

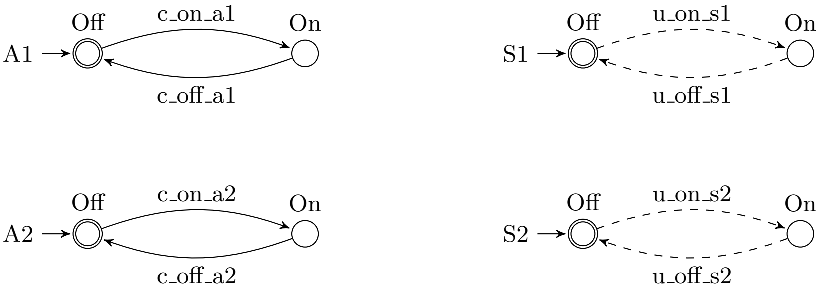


Fig. 9. An alternative model of the buffer.

the buffer to the next work station, a sensor S1 located at the entrance of the buffer to measure whether the buffer is full, and a sensor S2 located at the exit of the buffer to measure whether the buffer is empty. The component models are shown in Figure 9.

Several differences can be observed between the component models in Figures 8 and 9. First, in the high-level perspective model the machines in the proximity of the buffer are modeled, while in the input-output perspective actuators responsible for the movement of products are modeled. Secondly, the events in the high-level perspective have a complex meaning, like u_f_m1 representing that machine 1 has finished production and the product is placed in the buffer. And third, maybe the most important observation is that all component models in the high-level perspective are connected by shared events, while all component models in the input-output perspective are asynchronous and form a product system.

Requirements are formulated that express that the buffer may not overflow or underflow. Requirements R_1 and R_2 below are formulated for the high-level perspective model and requirements R_3 and R_4 for the input-output perspective model.

$$\begin{aligned} R_1: &\quad \text{c\_f\_m1 } \textbf{needs} \text{ B.Zero} \\ R_2: &\quad \text{c\_s\_m2 } \textbf{needs} \text{ B.One} \\[1em] R_3: &\quad \text{c\_on\_a1 } \textbf{needs} \text{ S1.Off} \\ R_4: &\quad \text{c\_on\_a2 } \textbf{needs} \text{ S2.On} \end{aligned}$$

While these requirements are very similar in form, the input-output perspective model and its requirements satisfy the Controllable and Nonblocking Modular Supervisors Properties as presented in [9]. Therefore, no synthesis is needed for this model and the component and requirement models are together already modular supervisors.

Waterway traffic light

The second example is a traffic light from a waterway lock, inspired by [28]. Such a traffic light is used to communicate with vessels whether they are allowed to enter the lock. The traffic light consists of three lamps, see Figure 10: a red one, a green one, and another red one. Four aspects, i.e., combinations of lamps turned on, have the following legal meaning in the communication with vessels.

- *Double red aspect.* This aspect is formed by having both red lamps on and the green lamp off. It indicates that the lock is out-of-service.
- *Red aspect.* This aspect is formed by having the top red lamp on and the green and bottom red lamps off. It indicates that vessels are not allowed to enter the lock from this side of the waterway.
- *Red-green aspect.* This aspect is formed by having the top red and green lamps on and the bottom red lamp off. It indicates to vessels that they may enter the lock soon, so captains should prepare their vessels.
- *Green aspect.* This aspect is formed by having the green lamp on and both red lamps off. It indicates that vessels are allowed to enter the lock.

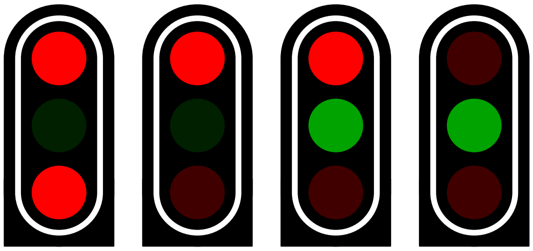


Fig. 10. The aspects of the lock traffic light: double red, red, red-green, and green.

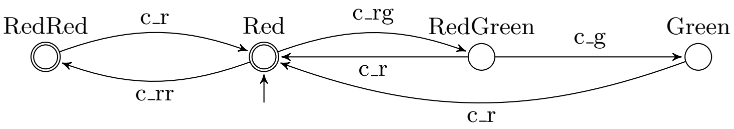


Fig. 11. The model of the traffic light as proposed in [28].

Figure 11 shows the model of the traffic light as proposed in [28]. It uses the four aspects of the traffic light as states and defines possible transitions between them. The events on the transitions do not correspond directly to a value change in one of the input or output signals of the control hardware.

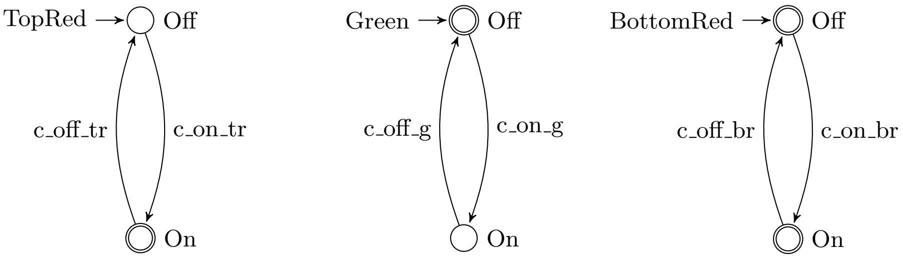


Fig. 12. The models of the traffic light using the input-output perspective.

Figure 12 shows the models of the traffic light when the input-output perspective is followed. Each lamp in the traffic light can be actuated separately, resulting in three asynchronous models TopRed, Green, and BottomRed. Each event now relates to a value change in the output signal of the controller hardware.

Several differences can be observed between the models in Figure 11 and the models in Figure 12. First, each model created with the input-output perspective is smaller than the single model with the aspect perspective, which makes understanding the model easier. Second, the three models with the input-output perspective are indeed a product system, so benefits in performance of synthesis as discussed in Section 3 also apply in this case. Third, the plant model with the input-output perspective describes more behavior, as it includes also illegal aspects of the traffic light. Therefore, the modeler needs explicitly exclude these illegal aspects with requirement models. These illegal aspects (and transitions to these aspects) have already been removed in the model in Figure 11.

In this particular example, there exists a injective mapping between states of the model in Figure 11 and states in the models in Figure 12. For example, the state RedRed maps to state TopRed.On, Green.Off, and BottomRed.On; and the steed Green maps to RedRed.Off, Green.On, and BottomRed.Off. Mappings for states Red and RedGreen can be derived similarly.

Wafer scanner logistics

The third example is a model of the wafer logistics in a lithography scanner, see [31]. A lithography scanner exposes silicon wafers to manufacture integrated circuits. Besides exposing a wafer, several pre- and postprocessing steps are performed in a lithography scanner, such as conditioning, aligning and measuring. These processing steps are performed multiple times before the integrated circuits on the wafer are finished. The goal of the supervisory controller is to properly manage the wafer logistics in such a scanner.

The wafer scanner logistics model of [31] deploys a product-based modeling perspective, where the products that go through the manufacturing process are modeled, as well as all actions that are possible on the products. Figure 13 shows two of the component models of the wafer scanner logistics. The model

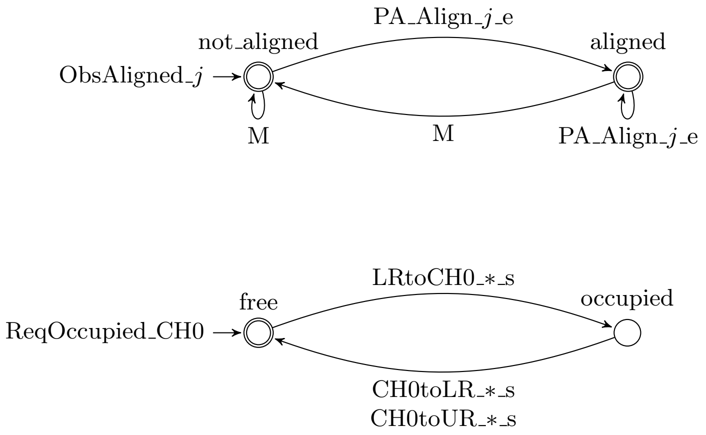


Fig. 13. The model ObsAligned_j of the alignment status of wafer j (left) and the model ReqOccupied_CH0 of the occupation of resource CH0 (right), both taken from [31].

ObsAligned_j models for each wafer $j \in J$ in the system, with J the set of all wafers, whether it is aligned or not. This component model represents a property of a product in the system. The model ReqOccupied_CH0 keeps track whether the resource CH0 is occupied by a wafer or not. As each wafer $j \in J$ may occupy this resource, this automaton needs to be able to synchronize with events from all wafers, which is in short denoted by $*$ in the model, e.g., LRtoCH0_$*$_s $= \{$LRtoCH0_j_s $\mid j \in J\}$. This model represents a property of a resource in the system.

The product-based modeling perspective results in all component models being connected, as they need to synchronize in shared events to track the different products through the system. This is detrimental to the applicability of synthesis, as mentioned in [30], since synthesizing a monolithic supervisor was not possible. Also, due to these strongly connected component models, modular and multilevel synthesis will not ease synthesis, as for each (group of) requirement(s) all component models need to be taken into account during synthesis.

In the PhD thesis [30] the explicit models of the products are removed and the remaining component models of the resources rewritten. This means that the model ObsAligned_j from Figure 13 is no longer included in the adapted model. The model ReqOccupied_CH0 from Figure 13 of resource CH0 is rewritten into CH0, as shown in Figure 14. The events related to each wafer j are replaced by generalized events. Furthermore, the events CH0_Measure and CH0_Expose on the self-loop in location occupied originated from another component model, not shown in Figure 13, which is removed in the adapted model. Now, for the adapted model a monolithic supervisor can be synthesized having 2190 states

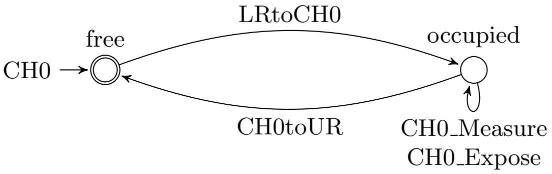


Fig. 14. The model CH0 of resource CH0, taken from [31].

and 6969 transitions. This is a significant synthesis performance increase, as no supervisor could be synthesized for the product-based perspective model.

6 Conclusion and future work

This paper presents three guidelines for modeling systems for which a supervisory controller needs to be synthesized. The first one expresses that independent plant components should be modeled as asynchronous plant models. The second one recommend that physical relationships between component models can be easily expressed with extended finite automata. The third one expresses that the input-output perspective of the control hardware should be used for the plant models. Examples from practice show how the guidelines can be used and that they can result in a considerable increase in performance of supervisory control synthesis.

Acknowledgments The authors thank Maria Angenent, Bert van der Vegt, and Han Vogel from Rijkswaterstaat for their feedback on the results.

References

1. Balemi, S.: Control of Discrete Event Systems: Theory and Application. Ph.D. thesis, Swiss Federal Institue of Technology Zurich, Zurich (1992)
2. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer, Boston, 2nd edn. (2008)
3. Fabian, M., Hellgren, A.: PLC-based implementation of supervisory control for discrete event systems. In: 37th IEEE Conference on Decision and Control. vol. 3, pp. 3305–3310 (1998). https://doi.org/10.1109/CDC.1998.758209
4. Fabian, M., Fei, Z., Miremadi, S., Lennartson, B., Åkesson, K.: Supervisory control of manufacturing systems using extended finite automata. In: Campos, J., Seatzo, C., Xie, X. (eds.) Formal Methods in Manufacturing, pp. 295–314. Industrial Information Technology, Taylor & Francis Inc. (2014)
5. Feng, L., Wonham, W.M.: Nonblocking coordination of discrete-event systems by control-flow nets. In: 46th IEEE Conference on Decision and Control. pp. 3375–3380. https://doi.org/10.1109/CDC.2007.4434160
6. Flordal, H., Malik, R.: Compositional verification in supervisory control. SIAM Journal on Control and Optimization **48**(3), 1914–1938. https://doi.org/10.1137/070695526

7. Göbe, F., Ney, O., Kowalewski, S.: Reusability and modularity of safety specifications for supervisory control. In: 21st IEEE International Conference on Emerging Technologies and Factory Automation. pp. 1–8 (2016). https://doi.org/10.1109/ETFA.2016.7733498
8. Gonzalez, A.G.C., Alves, M.V.S., Viana, G.S., Carvalho, L.K., Basilio, J.C.: Supervisory Control-Based Navigation Architecture: A New Framework for Autonomous Robots in Industry 4.0 Environments. IEEE Transactions on Industrial Informatics **14**(4), 1732–1743 (2018). https://doi.org/10.1109/TII.2017.2788079
9. Goorden, M.A., Fabian, M.: No synthesis needed, we are alright already. In: 15th IEEE International Conference on Automation Science and Engineering. pp. 195–202. https://doi.org/10.1109/COASE.2019.8843071
10. Goorden, M.A., van de Mortel-Fronczak, J.M., Etman, L.F.P., Rooda, J.E.: DSM-based analysis for the recognition of modeling errors in supervisory controller design. In: 21st International Dependency and Structure Modeling Conference. pp. 127–135 (2019). https://doi.org/10.35199/dsm2019.7
11. Goorden, M.A., van de Mortel-Fronczak, J.M., Reniers, M.A., Fokkink, W.J., Rooda, J.E.: The impact of requirement splitting on the efficiency of supervisory control synthesis. In: Larsen, K.G., Willemse, T. (eds.) Formal Methods for Industrial Critical Systems. Lecture Notes in Computer Science, vol. 11687, pp. 76–92. Springer. https://doi.org/10.1007/978-3-030-27008-7_5
12. Goorden, M.A., van de Mortel-Fronczak, J.M., Reniers, M.A., Fokkink, W.J., Rooda, J.E.: Structuring multilevel discrete-event systems with dependency structure matrices. IEEE Transactions on Automatic Control (2019). https://doi.org/10.1109/TAC.2019.292811, early access
13. Gössler, G., Sifakis, J.: Composition for component-based modeling. Science of Computer Programming **55**(1), 161–183. https://doi.org/10.1016/j.scico.2004.05.014
14. Grigorov, L., Butler, B.E., Cury, J.E.R., Rudie, K.: Conceptual design of discrete-event systems using templates. Discrete Event Dynamic Systems **21**(2), 257–303 (2011). https://doi.org/10.1007/s10626-010-0089-0
15. Komenda, J., Masopust, T., van Schuppen, J.H.: Control of an engineering-structured multilevel discrete-event system. In: 13th International Workshop on Discrete Event Systems. pp. 103–108 (2016)
16. Ma, C., Wonham, W.: Nonblocking Supervisory Control of State Tree Structures. No. 317 in Lecture Notes in Control and Information Sciences, Springer (2005)
17. Markovski, J., Jacobs, K.G.M., van Beek, D.A., Somers, L.J., Rooda, J.E.: Coordination of resources using generalized state-based requirements. In: 10th International Workshop on Discrete Event Systems. pp. 300–305 (2010)
18. Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional nonblocking verification of extended finite-state machines. Discrete Event Dynamic Systems **26**(1), 33–84 (2016). https://doi.org/10.1007/s10626-015-0217-y
19. Moormann, L., Maessen, P., Goorden, M.A., van de Mortel-Fronczak, J.M., Rooda, J.E.: Design of a tunnel supervisory controller using synthesis-based engineering (2020), accepted for ITA-AITES World Tunnel Congress
20. Ouedraogo, L., Kumar, R., Malik, R., Åkesson, K.: Nonblocking and safe control of discrete-event systems modeled as extended finite automata. IEEE Transactions on Automation Science and Engineering **8**(3), 560–569 (2011). https://doi.org/10.1109/TASE.2011.2124457
21. Pena, P.N., Cury, J.E.R., Lafortune, S.: Verification of nonconflict of supervisors using abstractions. IEEE Transactions on Automatic Control **54**(12), 2803–2815. https://doi.org/10.1109/TAC.2009.2031730

22. de Queiroz, M.H., Cury, J.E.R.: Modular supervisory control of large scale discrete event systems. In: Boel, R., Stremersch, G. (eds.) Discrete Event Systems, pp. 103–110. No. 569 in The Springer International Series in Engineering and Computer Science (2000)
23. Ramadge, P.J.G., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM Journal on Control and Optimization **25**(1), 206–230 (1987)
24. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE **77**(1), 81–98 (1989)
25. Ramos, A.L., Ferreira, J.V., Barceló, J.: Model-based systems engineering: An emerging approach for modern systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **42**(1), 101–111 (2012). https://doi.org/10.1109/TSMCC.2011.2106495
26. Reijnen, F.F.H., Erens, T.R., van de Mortel-Fronczak, J.M., Rooda, J.E.: Supervisory control synthesis for safety PLCs (2020), submitted to International Workshop on Discrete Event Systems
27. Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., Reniers, M.A., Rooda, J.E.: Application of dependency structure matrices and multilevel synthesis to a production line. In: 2nd IEEE Conference on Control Technology and Applications. pp. 458–464 (2018). https://doi.org/10.1109/CCTA.2018.8511449
28. Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., Rooda, J.E.: Supervisory control synthesis for a waterway lock. In: 1st IEEE Conference on Control Technology and Applications. pp. 1562–1568 (2017). https://doi.org/10.1109/CCTA.2017.8062679
29. Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., Rooda, J.E.: Supervisory control synthesis for a lock-bridge combination (2019), submitted to Discrete Event Dynamic Systems
30. van der Sanden, L.J.: Performance analysis and optimization of supervisory controllers. Ph.D. thesis, Eindhoven University of Technology (2018)
31. van der Sanden, L.J., Reniers, M.A., Geilen, M.C.W., Basten, T., Jacobs, J., Voeten, J.P.M., Schiffelers, R.R.H.: Modular model-based supervisory controller design for wafer logistics in lithography machines. In: 18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (2015)
32. Skoldstam, M., Åkesson, K., Fabian, M.: Modeling of discrete event systems using finite automata with variables. In: 46th IEEE Conference on Decision and Control. pp. 3387–3392 (2007). https://doi.org/10.1109/CDC.2007.4434894
33. Su, R., van Schuppen, J.H., Rooda, J.E.: Synthesize nonblocking distributed supervisors with coordinators. In: 17th Mediterranean Conference on Control and Automation. pp. 1108–1113 (2009). https://doi.org/10.1109/MED.2009.5164694
34. Swartjes, L., van Beek, D.A., Fokkink, W.J., van Eekelen, J.A.W.M.: Model-based design of supervisory controllers for baggage handling systems. Simulation Modelling Practice and Theory **78**, 28–50 (2017). https://doi.org/10.1016/j.simpat.2017.08.005
35. Swartjes, L.: Model-Based Design of Baggage Handling Systems. Ph.D. thesis, Eindhoven University of Technology (2018)
36. Theunissen, R.J.M., Petreczky, M., Schiffelers, R.R.H., Beek, D.A.v., Rooda, J.E.: Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. IEEE Transactions on Automation Science and Engineering **11**(1), 20–32 (2013)
37. Wonham, W.M., Ramadge, P.J.G.: Modular supervisory control of discrete-event systems. Mathematics of Control, Signals and Systems **1**(1), 13–30 (1988)

38. Wonham, W.M., Cai, K.: Supervisory Control of Discrete-Event Systems. Springer, 1st edn. (2019)
39. Zaytoon, J., Carre-Meneatrier, V.: Synthesis of control implementation for discrete manufacturing systems. International Journal of Production Research **39**(2), 329–345 (2001). https://doi.org/10.1080/00207540010002388