The Impact of Requirement Splitting on the Efficiency of Supervisory Control Synthesis*

Martijn Goorden[1], Joanna van de Mortel-Fronczak[1], Michel Reniers[1], Wan Fokkink[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 for a cyber-physical system based on models of the uncontrolled system components and models of the control requirements. Although several synthesis procedures have been proposed and automated, obtaining correct and useful models of industrial-size applications that are needed as their input remains a challenge. We show that the efficiency of supervisor synthesis techniques tends to increase significantly if a single large requirement is split into a set of smaller requirements. A theoretical underpinning is provided for showing the strength of this modeling guideline. Moreover, several examples from the literature as well as some real-life case studies are included for illustration.

Keywords: Supervisory control synthesis · Automata · Requirements engineering.

1 Introduction

The design of supervisors for cyber-physical systems has become a challenge as they 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 for these systems. Model-based systems engineering methods can help in overcoming these difficulties, see [23].

For the design of supervisors, the supervisory control theory of Ramadge-Wonham [21, 22] provides means to synthesize supervisors from a model of the uncontrolled plant (describing what the system *could* do) and a model of the control requirements (describing what the system *should* 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.

A major drawback of synthesizing monolithic supervisors is its computational complexity, both in the time and memory domain. Although the time complexity of this step is polynomial in the number of states that represent the system, this number increases exponentially with the number of constituent models of the different components in the system, as already observed in [22]. For industrial systems, the number of states can easily reach an order of 10^{100} states. Different supervisor architectures are exploited in an attempt to overcome these computational difficulties: modular [20], hierarchical [34], decentralized [28], distributed [3], multilevel [12], and compositional supervisory control synthesis [18]. Modular, decentralized, and multilevel synthesis are closely related and in this paper we refer to them as module-based supervisor architectures.

While these architectures claim to gain computational efficiency, in practice the observed gain depends on the models provided as input for these synthesis algorithms. Moreover, as systems can be modeled in several ways, i.e., there is not a single correct model formulation for a certain plant and its requirements, an engineer might model an industrial system in a disadvantageous way and might (wrongly) conclude that supervisory control synthesis is not possible for his system.

To the best of our knowledge, not much attention has been paid in the literature to the fact that the way in which models are defined can be of a significant influence on the efficiency of the synthesis procedure. A notable exception is [11], where symmetry in the model is exploited to efficiently synthesize a supervisor. Others [6, 7, 10] 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., the obtained models exhibit the behavior the engineer intended to model.

The purpose of this paper is to provide a modeling guideline to (re)formulate the models such that the applicability of supervisory control synthesis techniques increases. This modeling guideline concerns the modeling of the requirements and expresses that they should be split into smaller ones when possible. We show theoretically why this modeling guideline increases the applicability of supervisory control synthesis. Essentially, smaller requirements allow module-based synthesis techniques to solve numerous but computationally easier problems instead of those obtained with large requirements, because each new requirement relates to fewer plant models than the original large requirement. For multilevel synthesis, this effect is visualized by displaying the dependencies with a Dependency Structure Matrix, see [5]. Experimental results of several case studies show that this efficiency gain can indeed be obtained in practice. By proposing this guideline and by providing examples, our aim is to assist practitioners in applying supervisory control synthesis.

Requirement specifications in practice often violate the aforementioned guideline, which turns out to be detrimental for supervisory control synthesis. Although the guideline may sound somewhat obvious, it required several real-life case studies with supervisory control synthesis to grasp its importance [25–27]. These case studies were performed in the context of a research project with Rijkswaterstaat, the national organisation responsible for the main infrastructure like roads and bridges in the Netherlands. Notably, the so-called Oisterwijksebaan revolving bridge in the Dutch city of Tilburg was recently operated by PLC code automatically generated from the requirements, by means of the CIF supervisory control toolset [2]. These case studies have inspired us to formulate several modeling guidelines. The aim of this paper is to describe one of them in detail.

The paper is structured as follows. Section 2 provides the preliminaries of this paper. Section 3 continues by discussing the guideline concerning the model of the requirement in detail including a theoretical substantiation. In Section 4, the guideline is demonstrated with an example of supervisory control for an infrastructural system. Section 5 provides experimental results with cases also from other application domains where applying the guideline benefits supervisory control synthesis. 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 [4, 33].

2.1 Automata

An automaton is a five-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the (finite) state set, Σ is the alphabet of events, $\delta : Q \times \Sigma \to Q$ 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 string. 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.

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*

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

where

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

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 $\| \mathcal{G}$, is defined as $\| \mathcal{G} = G_1 \| \ldots \| G_m$, and the synchronous composition of two composed systems $\mathcal{G}_1 \| \mathcal{G}_2$ is defined as $(\| \mathcal{G}_1) \| (\| \mathcal{G}_2)$. 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 $i,j \in [1,m], i \neq j$ [22].

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 $\delta_K(q_{0,K},su)!$.

2.2 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 [4,21,22,33]. 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 $P \| S$ should always satisfy the imposed requirements, i.e., $\mathcal{L}(P \| S) \subseteq \mathcal{L}(P \| R)$
– *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., $P \| S$ 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 $\mathcal{L}(P \| S') \subseteq \mathcal{L}(P \| S)$.

For the purpose of supervisor synthesis, requirements can be modeled with automata and state-based expressions, as introduced in [15, 16]. The latter is useful in practice, as engineers tend to formulate requirements based on states of the plant. To refer to states of the plant, we introduce the notation $P.q$ which refers to state q of plant P. State references can be combined with the Boolean literals **T** and **F** and logic connectives to create predicates.

A state-event invariant expression formulates conditions on the enablement of an event based on states of the plant, i.e., the condition should evaluate to true for the event to be enabled. A state-event invariant expression is of the form σ **needs** C where σ is an event and C a predicate stating the condition. Let R be a state-event invariant expression, then $\mathit{event}(R)$ returns the event used in R and $\mathit{cond}(R)$ returns the condition predicate. An example of a state-event invariant expression is a **needs** $P_1.q_1 \wedge P_2.q_2$ formulating that event a is only allowed when automaton P_1 is in state q_1 and automaton P_2 is in state q_2.

Given a composed system representation of the plant $P_s = \{P_1, \ldots, P_m\}$ and a collection of requirements $R_s = \{R_1, \ldots, R_n\}$, we define the tuple (P_s, R_s) as the *control problem* for which we want to synthesize a supervisor.

Monolithic supervisory control synthesis results in a single supervisor S from a single plant model and a single requirement model [21]. 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 P_s and R_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 R_s [32]. 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 (P_s, R_s) with $R_s = \{R_1, \ldots, R_n\}$, modular synthesis solves n control problems $(P_s, \{R_1\}), \ldots, (P_s, \{R_n\})$. Each control problem $(P_s, \{R_i\})$ for $i \in [1, n]$ results in a safe, controllable, nonblocking, and maximally permissive supervisor S_i. Unfortunately, the collection of supervisors $S_s = \{S_1, \ldots, S_n\}$ can be conflicting, i.e., $S_1 \parallel \ldots \parallel S_n$ can be blocking. A nonconflicting check can verify whether S_s is nonconflicting, see [19,30]. In the case that S_s is nonconflicting, S_s is also safe, controllable, nonblocking, and maximally permissive for the original control problem (P_s, R_s) [32]. In the case that S_s is conflicting, an additional coordinator C can be synthesized such that $S_s \cup \{C\}$ is safe, controllable, nonblocking, and maximally permissive for the original control problem (P_s, R_s) [29]. An extension to this approach, as proposed by [20], states that instead of synthesizing each time with the complete plant P_s, 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.

Decentralized supervisory control synthesis has a similar setting as modular supervisory control synthesis, except that each supervisor is only allowed to observe certain events, called local events, instead of all events [14]. This results in the notion of observability, which is not further discussed in this paper. Nevertheless, also for decentralized supervisory control synthesis with multiple requirements, the obtained supervisors may be conflicting.

Multilevel supervisory control synthesis is inspired by decompositions of systems by engineers [12]. 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 using those requirements and the plant model representing the subsystem. Again, the collection of synthesized supervisors may be conflicting.

3 Modeling guideline and theoretical substantiation

When formulating the requirements, engineers often tend to think in desired control logic and formulate this logic as requirements. The benefit of supervisory control synthesis is that an engineer is able to focus on *what* the system should do, not *how* it should do it. By shifting from specifying how to specifying what, requirements do not always become smaller. In this section, we show that module-based supervisor architectures benefit from having small requirement models.

We specifically focus on requirements formulated with state-event invariant expressions. This form matches well with requirements formulated in a natural language like, e.g., English, see [16]. Furthermore, requirements for industrial-size applications often originate from failure-mode analysis [17]. States are identified in which some actuator actions would result in unsafe behavior. Therefore, this form is frequently used in real-life case studies of infrastructural systems, see [25–27].

The modeling guideline is formulated as follows:

Split requirements formulated with state-event invariant expressions into a set of smaller ones.

Splitting a state-event invariant expression can be done as follows. Consider requirement σ **needs** C expressing that event σ is only allowed when condition C holds. When this condition is denoted in conjunctive normal form, i.e., $C = C_1 \wedge \ldots \wedge C_l$, the single requirement can be split into multiple requirements σ **needs** $C_1, \ldots, \sigma$ **needs** C_l. Due to the safety property of synthesized supervisors, mentioned in Section 2.2, the set of requirements is equivalent to the single requirement. In the rest of this section, we show the benefit of having small requirements theoretically.

Splitting requirements in the form of propositional formula to benefit controller synthesis is a well-known strategy for software product lines, see for example [1, 9]. Here, a requirement, called a feature constraint, is split into several configurations (or products) each describing a specific feature combination. For each configuration a controller is synthesized. There are two main differences between that work and the work in this paper. First, a feature constraint limits the possible configurations, while requirements in this paper limit the behavior of one configuration. Second, only one of the synthesized supervisors for a software product line is active (the one for that specific configuration), while in this work all modular supervisors work in conjunction.

3.1 Theoretical substantiation

Consider the plant being modeled with a product system $P_s = \{P_1, \ldots, P_n\}$, and assume that a requirement R may also be modeled with a set of requirements $R_s = \{R_1, \ldots, R_m\}$ such that $R = \parallel R_s$.[3] Any module-based supervisor architecture ensures that for each (set of) requirement(s) synthesis is performed with only those plant models that are related to the (set of) requirement(s). Reformulating a larger requirement into smaller requirements ensures that module-based supervisor architectures can identify smaller control problems to solve. Hence, a reduction in computational effort is gained.

For modular supervisory control synthesis, the analysis above can be even further detailed as follows. Assume for simplicity that requirement R relates to *all* plant models in P_s, while each smaller requirement $R_j \in R_s$ only refers to a subset $P_{s,j} \subseteq P_s$. In the case of a single requirement R, modular supervisory control synthesis obtains a supervisor for control problem $(P_s, \{R\})$. In the case of multiple smaller requirements, m supervisors are obtained for each control problem $(P_{s,j}, \{R_j\}), 1 \leq j \leq m$. As $|P_{s,j}| \leq |P_s|$ holds, the state-space size of $P_{s,j}$ is smaller or equal than P_s. The computational effort for each synthesis problem is therefore at most equal to that of monolithic synthesis. Yet, m supervisors are synthesized instead of just one, so there is a tradeoff between more control problems to solve and creating smaller control problems to solve. As the state-space size grows exponentially with the number of automata, reducing the number of plant components often has a larger effect than synthesizing more supervisors. Experimental results in Section 5 confirm this tradeoff.

For multilevel supervisory control synthesis, we analyze the effect of splitting requirements differently than for modular supervisory control synthesis. In multilevel synthesis, the system is decomposed into subsystems. The dependencies between plant models indicate how the system may be decomposed. For the purpose of multilevel synthesis, analyzing the dependencies between plant models induced by the requirement models is valuable, see [8]. Dependencies between two plant models can be formalized as follows. Given $P_i, P_j \in P_s, P_i \neq P_j$, there is a dependency between P_i and P_j if and only if there exists a requirement $R_k \in R_s$ such that both plant models are used in R_k. A plant model is used in a state-event invariant expression if the event in the requirement originates from the alphabet of that plant model or the condition uses a state of that plant model. For example, in $R = P_1.\sigma$ **needs** $P_2.q_2$, where we used the notation $P_1.\sigma$ to indicate that σ is in the alphabet of P_1, plant models P_1 and P_2 are used in R.

Now, consider requirement $R = P.\sigma$ **needs** C where condition C is the conjunction of some state references, that is $C = P_1.q_1 \wedge \ldots \wedge P_l.q_l$. This requirement results in dependencies between plant models P and P_1, P and P_2, and so on, and also in dependencies between any pair $(P_i, P_j), i, j \in [1, l], i \neq j$. These

[3] Here we have a slight abuse of notation of the synchronous product operator, as this one is only formally defined for automata. In case of two requirements modeled with state-event invariant expressions restricting the same event σ, denoted by $R_i = \sigma$ **needs** $C_1, i \in \{1, 2\}$, we define $R_1 \parallel R_2 = \sigma$ **needs** $C_1 \wedge C_2$.

D	P_1	P_2	P_3	P_4	P
P_1	-	1	1	1	1
P_2	1	-	1	1	1
P_3	1	1	-	1	1
P_4	1	1	1	-	1
P	1	1	1	1	-

D'	P_1	P_2	P_3	P_4	P
P_1	-				1
P_2		-			1
P_3			-		1
P_4				-	1
P	1	1	1	1	-

Fig. 1. Left the DSM D constructed with the original requirement R and right the DSM D' with the set of splitted requirements R_s.

dependencies can be visualized with a Dependency Structure Matrix (DSM), see [5]. Figure 1 shows on the left the DSM D for requirement R with $l = 4$. A dependency between plant models is indicated in this DSM with a 1, no dependency is indicated with an empty cell. Such a visualization shows that all plant models are related with each other. Therefore, multilevel synthesis considers plant models $P, P_1, \ldots, P_l$ as a single subsystem and synthesizes a supervisor for control problem $(\{P, P_1, \ldots, P_l\}, \{R\})$.

When requirement R is split into multiple requirements collected in set $R_s = \{R_1, \ldots, R_l\}$ where $R_k = P.\sigma$ **needs** $P_k.q_k, k \in [0, l]$, the dependencies between the plant models reduces. There are still dependencies between plant models P and P_1, P and P_2, and so on till P and P_l, yet there are no longer dependencies between any pair $(P_i, P_j), i, j \in [1, l], i \neq j$, which is the case with the single requirement R. The effect of splitting requirements is visualized in DSM D' in Figure 1. The number of dependencies has reduced significantly. This reduction allows multilevel synthesis to decompose the system into smaller subsystems, for example into two subsystems where the first is composed of plant models P, P_1, P_2 and the second of plant models P, P_3, P_4. Similar to modular synthesis, smaller subsystems result in smaller control problems to solve, resulting in a reduction of computational effort. Therefore, splitting requirements can be beneficial for multilevel supervisory control synthesis.

3.2 Conflicting supervisors

Similar to modular synthesis, splitting requirements introduces an over-approximation. Synthesizing multiple supervisors for the split requirements may result in conflicting supervisors.

Consider the following example to illustrate the over-approximation induced by splitting requirements. Figure 2 shows the plant models of a door actuator and a door sensor. Requirement R = A_Door.c_off **needs** S_Door.Off $\wedge$ S_Door.On expresses that the actuator may only be turned off when the door sensor is off and on. This requirement can be split into the two requirements R_1 = A_Door.c_off **needs** S_Door.Off and R_2 = A_Door.c_off **needs** S_Door.On. Since an automaton cannot be in two locations at the same time, the condition of the

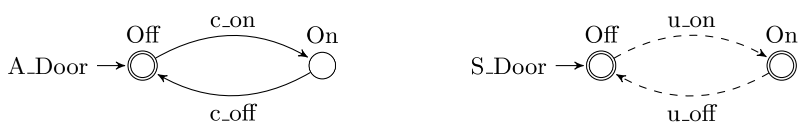

Fig. 2. Examples of two plant models, with an actuator of a door and a sensor of a door. Concentric circles indicate marked locations. Solid arrows indicate controllable events while dashed arrows indicate uncontrollable events.

original requirement R can never be satisfied, effectively disabling event c_off indefinitely. A supervisor synthesized for the single requirement disables event c_on of the actuator, because location On is not marked. When the single requirement R is replaced by the two requirements R_1 and R_2, conflicting modular supervisors are synthesized. Each local supervisor will not disable event c_on, allowing the actuator to block in location On.

In general, one can perform a nonconflicting check after synthesizing modular or multilevel supervisors for the split requirements. Yet, as discussed in Section 2.2, a nonconflicting check should always be performed if modular or multilevel synthesis is applied, even when requirements are not split. It is an interesting question for future research to determine the effect of splitting requirements on the efficiency of the nonconflicting check and on the synthesis of a coordinator.

The example may indicate that splitting 'bad' requirements could induce conflicts. A requirement demanding an automaton to be in multiple states at the same time would probably not be the intention of an engineer. Yet, there is no guarantee that an engineer does not formulate such a requirement. Notwithstanding the general situation, the following conjecture formalizes the situations encountered in cases where requirements can be split which will not introduce conflicting problems.

Conjecture 1. Let $\mathcal{P} = \{P, P_1, \ldots, P_m\}$ be a product system and requirement $R = P.\sigma$ **needs** $C_1 \wedge C_2 \wedge \ldots \wedge C_n$ such that no pair of conditions $C_i, C_j, i, j \in [1, n], i \neq j$ uses the same plant model. Construct the set of split requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$ with each split requirement being $R_i = P.\sigma$ **needs** C_i. Then the set of modular supervisors for $\mathcal{R}$ is nonconflicting.

4 Demonstration with case study of infrastructural system

Splitting state-event invariant requirements is demonstrated with the model of Lock III, located at Tilburg, The Netherlands. Figure 3 shows the lock. The model of Lock III is given in [25]. A lock is an infrastructural system in rivers and canals with the purpose to maintain different water levels outside the lock while also allowing the vessels to pass from one level to the other. A lock consists primarily of a lock chamber with a lock head on each side. The main subsystems

Fig. 3. Photo of Lock III, located at Tilburg, The Netherlands. Image from https://beeldbank.rws.nl, Rijkswaterstaat.

of a lock head are the gates, water leveling systems, and the incoming and outgoing traffic lights. Supervisory control is deployed to ensure safe operation of the system. In this context, safety not only concerns avoiding human injuries or causalities, but also water management as large parts of The Netherlands are located below water level.

For modeling convenience, there is also the state-event invariant expression D **disables** σ, which expresses that event σ is disabled when condition D holds. This expression has the same expressiveness as the form σ **needs** C: D **disables** σ is equivalent to σ **needs** $\neg D$. Following the same splitting mechanism as introduced with the guideline, requirements of the form D **disables** σ can be split if condition D is in disjunctive normal form, i.e., $D = D_1 \vee \ldots \vee D_k$.

The guideline is demonstrated with the following requirement: it is unsafe to open a gate if (1) the water-leveling system at the other side is not closed, or (2) the gate at the other side is not closed, or (3) there is no equal water over the gate, or (4) the incoming traffic light at that lock head is not showing a red or red-red aspect, or (5) the outgoing traffic light at that lock head is not showing a red aspect. For one of the gates this requirement is formalized in the model as

$$
\begin{aligned}
&(1)\ \text{culvert\_N.S.flow} \vee \text{culvert\_N.A.open} \vee \text{culvert\_S.S.flow} \vee \text{culvert\_S.A.open} \vee \\
&(2)\ \neg\text{gate\_U\_N.S.closed} \vee \text{gate\_U\_N.Dir.opening} \vee \\
&\qquad \neg\text{gate\_U\_S.S.closed} \vee \text{gate\_U\_S.Dir.opening} \vee \\
&(3)\ \text{s\_equal\_D.off} \vee
\end{aligned}
$$

(4) ¬(in_D_N.S.red ∨ in_D_N.S.redred) ∨ ¬(in_D_N.A.red ∨ in_D_N.A.redred) ∨
¬(in_D_S.S.red ∨ in_D_S.S.redred) ∨ ¬(in_D_S.A.red ∨ in_D_S.A.redred) ∨
(5) ¬out_D_N.S.red ∨ ¬out_D_N.A.red ∨ ¬out_D_S.S.red ∨ ¬out_D_S.A.red
disables gate_D_N.c_open,

where before the first full stop (.) in every state and event name the letter D is an abbreviation for downstream, U for upstream, N for north, and S for south, and where after the first full stop the letter A stands for actuator and S for sensor. The five unsafe situations in which the gate should not open are indicated in the requirement.

The first option for splitting this requirement is creating five requirements, one for each unsafe situation. This results in the following five requirements:

(1) culvert_N.S.flow ∨ culvert_N.A.open ∨ culvert_S.S.flow ∨ culvert_S.A.open
disables gate_D_N.c_open,
(2) ¬gate_U_N.S.closed ∨ gate_U_N.Dir.opening ∨ ¬gate_U_S.S.closed ∨
gate_U_S.Dir.opening
disables gate_D_N.c_open,
(3) s_equal_D.off
disables gate_D_N.c_open,
(4) ¬(in_D_N.S.red ∨ in_D_N.S.redred) ∨ ¬(in_D_N.A.red ∨ in_D_N.A.redred) ∨
¬(in_D_S.S.red ∨ in_D_S.S.redred) ∨ ¬(in_D_S.A.red ∨ in_D_S.A.redred)
disables gate_D_N.c_open,
(5) ¬out_D_N.S.red ∨ ¬out_D_N.A.red ∨ ¬out_D_S.S.red ∨ ¬out_D_S.A.red
disables gate_D_N.c_open.

By specifying these five requirements instead of one, the readability and maintainability of the models also increases. Yet, these requirements can be split even further, as each condition is still in disjunctive normal form. Hence, 17 requirements can be formulated, of which the first four originated from (1) are

(1*a*) culvert_N.S.flow **disables** gate_D_N.c_open,
(1*b*) culvert_N.A.open **disables** gate_D_N.c_open,
(1*c*) culvert_S.S.flow **disables** gate_D_N.c_open,
(1*d*) culvert_S.A.open **disables** gate_D_N.c_open.

The other requirements can be split similarly.

Another requirement describes normal closing of a gate and expresses that a gate may only be closed if (1) the command to close the gate is given, and (2) the gate is not yet closed, and (3) the command to stop the gate is not given. The model of this textual requirement for one of the gates is

gate_D_N.c_close **needs** cmd_D_gate_close ∧ ¬gate_D_N.S.closed ∧
¬cmd_stop_D_gate,

Table 1. Experimental results for synthesizing modular and multilevel supervisors with the original and adapted Lock III models. The reported state-space size for modular and multilevel synthesis is the sum of the state-space sizes of the individual supervisors. The number of supervisors refers to the result of multilevel synthesis, monolithic synthesis results in only one supervisor and modular synthesis creates a supervisor for each requirement.

Model	Number of requirements	Monolithic	Modular	Multilevel	Number of supervisors
Original	142	$6.0 \cdot 10^{24}$	$1.60 \cdot 10^{13}$	$1.45 \cdot 10^{19}$	7
Adapted	358	$6.0 \cdot 10^{24}$	$1.32 \cdot 10^{05}$	$4.62 \cdot 10^{09}$	34

where D is an abbreviation for downstream, N for north, S for sensor, and cmd for command. The three terms of the condition are conjunctive, thus this requirement can be split into three smaller requirements as follows:

$$\begin{aligned}&\text{gate\_D\_N.c\_close } \textbf{needs} \text{ cmd\_D\_gate\_close},\\&\text{gate\_D\_N.c\_close } \textbf{needs} \ \neg\text{gate\_D\_N.S.closed},\\&\text{gate\_D\_N.c\_close } \textbf{needs} \ \neg\text{cmd\_stop\_D\_gate}.\end{aligned}$$

Finally, not all requirements may be split. Consider the requirement expressing that the outgoing traffic light may only switch to a red aspect if the command for showing the red aspect is given or any stop command is given. This requirement is formalized for one of the outgoing traffic lights as

$$\text{out\_D\_N.c\_red } \textbf{needs} \text{ cmd\_D\_out\_r} \vee \text{cmd\_stop}.$$

Experimental results are shown in Table 1. These results have been obtained with the CIF toolset [2] and the models can be accessed at a GitHub repository[4]. For both the original model and the adapted model we show the number of requirements, the controlled state-space size of the monolithic supervisor, the sum of the controlled state-space sizes of each modular and multilevel supervisor, and the number of multilevel supervisors. Splitting the requirements more than doubles the number of requirements and significantly increases the efficiency of both modular and multilevel supervisory control synthesis. Focussing on multilevel synthesis, the gain of using that supervisor architecture for the original model is already substantial comparing to monolithic synthesis. Yet, for the adapted model, the state-space size of the multilevel supervisors approaches the result of modular supervisors by synthesizing only 34 supervisors instead of 358 supervisors, respectively. Also, the number of multilevel supervisors indicates that by splitting the requirements the system can be decomposed into more subsystems, as what is expected from the analysis in Section 3.1.

[4] https://github.com/magoorden/SplittingRequirements

5 Four case studies with experimental results

In this section, the modeling guideline described in Section 3 is applied on several other models of real-life case studies. We first introduce the case studies and show a typical requirement that is split according to the modeling guideline. Subsequently, experimental results are shown after applying modular and multilevel supervisory control synthesis on these models.

5.1 Case studies description

Case **Marijke**. In this case study, the Prinses Marijke complex is modeled, see [26]. This infrastructural complex is located in the center of The Netherlands and consists of two waterway locks and a storm surge barrier. In case of high water levels in the Amsterdam-Rhine Canal, the barrier is closed and vessels need to use the waterway locks. In all other conditions, the barrier is opened and vessels can pass under it, without using the waterway locks.

The models of the locks in the Prinses Marijke complex are similar to the model of Lock III, see Section 3. Only the modeling level, or abstraction detail, differs. Therefore, the same requirements are specified, which opens the opportunity to split them.

Case **ADAS**. In this case study, an Advanced Driver Assistant Systems (ADAS) is modeled, see [13]. In such an application, a supervisor is synthesized to safely switch in a vehicle between the modes 'no cruise control (NCC)', 'cruise control (CC)', and 'adaptive cruise control (ACC)'. Based on input from the driver as well as vehicle sensors, the vehicle may or may not switch between these different modes of cruise control.

One of the formulated requirements is related to the desired behavior of the CC mode. It expresses that the set-point velocity can be decreased if CC is active **and** the brake sensor is off **and** the set-point velocity is higher than 30 km/h **and** the CC lever is pushed up for longer than 0.5 s **and** a set-point velocity is stored **and** CC is enabled **and** the vehicle velocity is higher than 30 km/h. This single requirement can be split into seven smaller requirements.

Case **FESTO**. In this case study, a production line designed by FESTO is modeled, see [24]. The FESTO production line is designed for vocational training in the field of industrial automation. While no real production takes place, all movements, velocities, and timings are as if it were. The production line consists of six workstations with in total 28 actuators, like DC motors and pneumatic cylinders, and 59 capacitive, optical, and inductive sensors.

In the first workstation, products enter the system from a storage tube. At the bottom of the tube, a pusher is able to push a product out. This pusher is only allowed to push (extend) if the system is initialized **and** the pusher is fully retracted **and** there is a product in the storage tube **and** the output place to push the product to is empty. This example requirement can be split into four smaller requirements formulating together the same desired behavior.

Table 2. Experimental results for synthesizing modular and multilevel supervisors with the original and adapted models of the several case studies. The reported state-space size for modular and multilevel synthesis is the sum of the state-space sizes of the individual supervisors. The number of supervisors refers to the result of multilevel synthesis, monolithic synthesis results in only one supervisor and modular synthesis creates a supervisor for each requirement.

Model	Variant	Number of requirements	Monolithic	Modular	Multilevel	Number of supervisors
LockIII	Original	142	$6.0 \cdot 10^{24}$	$1.60 \cdot 10^{13}$	$1.45 \cdot 10^{19}$	7
	Adapted	358	$6.0 \cdot 10^{24}$	$1.32 \cdot 10^{05}$	$4.62 \cdot 10^{09}$	34
Marijke	Original	248	$6.68 \cdot 10^{26}$	$1.29 \cdot 10^{7}$	$5.50 \cdot 10^{12}$	26
	Adapted	529	$6.68 \cdot 10^{26}$	$2.24 \cdot 10^{5}$	$4.03 \cdot 10^{11}$	33
ADAS	Original	33	$2.0 \cdot 10^{10}$	$1.5 \cdot 10^{4}$	$1.1 \cdot 10^{8}$	8
	Adapted	72	$2.0 \cdot 10^{10}$	$1.1 \cdot 10^{3}$	$5.2 \cdot 10^{5}$	16
FESTO	Original	78	$2.2 \cdot 10^{25}$	$2.10 \cdot 10^{4}$	$4.00 \cdot 10^{6}$	12
	Adapted	205	$2.2 \cdot 10^{25}$	$2.00 \cdot 10^{3}$	$5.06 \cdot 10^{4}$	24

5.2 Results

For each case study, requirements are split as much as possible according to the modeling guideline of Section 3, which results in the original model and an adapted model. Subsequently, monolithic, modular, and multilevel supervisory control synthesis are applying with the CIF toolset [2].

The results are shown in Table 2. For the three different synthesis techniques, the controlled state space is reported. For monolithic synthesis, the number is the state-space size of the single synthesized supervisor; for modular and multilevel synthesis, the number is the sum of the state-space sizes of the individual supervisors. The number of supervisors in the table refers to the number of supervisors of multilevel synthesis. The number of supervisors for modular synthesis equals the number of requirements and for monolithic synthesis there is only one supervisor. The results from Lock III, discussed in Section 4, are added for completeness.

For all four cases, adapting the models by splitting requirements increases the number of requirements significantly, it often more than doubles. The results for modular and multilevel synthesis indicate that splitting the requirements is beneficial for the efficiency of these supervisor architectures. For multilevel synthesis, splitting the requirements allows to decompose the system differently such that more subsystems are identified. Therefore, smaller control problems are defined to be solved, resulting in the reduction of the computational effort.

As expected, the obtained efficiency gain of splitting the requirements differs per model. For example, reformulating the model of Lock III allows multilevel synthesis to formulate an efficient decomposition, indicated by the state-space size and the number of supervisors, while the reduction is minimal for the model of the Prinses Marijke complex. Nevertheless, reformulating the model by splitting the requirements seems to be always valuable for models of real-life cases.

6 Conclusion and future work

This paper presents a guideline expressing that requirements should be split into smaller ones, each referring to less plant models than before. Theoretical substantiation is provided for the effectiveness of this guideline. Examples from practice show how the guideline can be used. Experimental results indicate that splitting requirements increases the applicability and efficiency of module-based supervisor architectures.

The examples indicate that automatic model transformation based on this guideline should be possible. Future work includes the design and implementation of such transformations. Furthermore, Section 3 showed an example of a requirement that could not be split. In [31], the introduction of new event in the plant is suggested to circumvent this issue. It is worth investigating this suggestion, albeit that also the plant model needs to be adapted. Finally, another direction for future research is considering requirements in the form of state invariant expressions, like the one expressing that actuators A and B may never be both on at the same time, and determining whether, for example, a logically equivalent set of state-event invariant expressions may be more beneficial for module-based supervisor architectures.

Acknowledgments The authors thank Ferdie Reijnen for providing the models of Lock III and the Prinses Marijke complex. The authors thank Rijkswaterstaat, part of the Dutch Ministry of Infrastructure and Water Management, for providing funding for this research. In particular, the authors thank Maria Angenent, Bert van der Vegt, and Han Vogel for their feedback on the results.

References

1. Basile, D., ter Beek, M.H., Di Giandomenico, F., Gnesi, S.: Orchestration of dynamic service product lines with featured modal contract automata. In: 21st International Systems and Software Product Line Conference - Volume B. pp. 117–122. ACM (2017). https://doi.org/10.1145/3109729.3109741
2. van Beek, D.A., Fokkink, W.J., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J.M., Reniers, M.A.: CIF 3: Model-based engineering of supervisory controllers. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 575–580. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (Apr 2014). https://doi.org/10.1007/978-3-642-54862-8_48
3. Cai, K., Wonham, W.M.: Supervisor localization: A top-down approach to distributed control of discrete-event systems. IEEE Transactions on Automatic Control **55**(3), 605–618 (Mar 2010)

4. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer, Boston, 2nd edn. (2008)
5. Eppinger, S.D., Browning, T.R.: Design structure matrix methods and applications. MIT press (2012)
6. 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., Boca Raton (Feb 2014)
7. Göbe, F., Ney, O., Kowalewski, S.: Reusability and modularity of safety specifications for supervisory control. In: IEEE 21st International Conference on Emerging Technologies and Factory Automation. pp. 1–8 (Sep 2016). https://doi.org/10.1109/ETFA.2016.7733498
8. Goorden, M.A., van de Mortel-Fronczak, J.M., Reniers, M.A., Rooda, J.E.: Structuring multilevel discrete-event systems with dependency structure matrices. In: 56th IEEE Conf. on Decision and Control. pp. 558–564 (Dec 2017). https://doi.org/10.1109/CDC.2017.8263721
9. Greenyer, J., Brenner, C., Cordy, M., Heymans, P., Gressi, E.: Incrementally synthesizing controllers from scenario-based product line specifications. In: 9th Joint Meeting on Foundations of Software Engineering. pp. 433–443. ACM (2013). https://doi.org/10.1145/2491411.2491445
10. 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 (Jun 2011). https://doi.org/10.1007/s10626-010-0089-0
11. Jiao, T., Gan, Y., Xiao, G., Wonham, W.M.: Exploiting symmetry of discrete-event systems by relabeling and reconfiguration. IEEE Transactions on Systems, Man, and Cybernetics: Systems pp. 1–12 (2018). https://doi.org/10.1109/TSMC.2018.2795011
12. Komenda, J., Masopust, T., van Schuppen, J.H.: Control of an engineering-structured multilevel discrete-event system. In: 13th Int. Workshop on Discrete Event Systems. pp. 103–108 (May 2016)
13. Korssen, T., Dolk, V., van de Mortel-Fronczak, J.M., Reniers, M.A., Heemels, M.: Systematic model-based design and implementation of supervisors for advanced driver assistance systems. IEEE Transactions on Intelligent Transportation Systems **19**(2), 533–544 (2017). https://doi.org/10.1109/TITS.2017.2776354
14. Lin, F., Wonham, W.M.: Decentralized control and coordination of discrete-event systems with partial observation. IEEE Transactions on Automatic Control **35**(12), 1330–1337 (Dec 1990). https://doi.org/10.1109/9.61009
15. Ma, C., Wonham, W.: Nonblocking Supervisory Control of State Tree Structures. No. 317 in Lecture Notes in Control and Information Sciences, Springer Berlin Heidelberg (2005)
16. Markovski, J., Jacobs, K.G.M., van Beek, D.A., Somers, L.J., Rooda, J.E.: Coordination of resources using generalized state-based requirements. pp. 300–305 (2010)
17. Modarres, M.: Risk Analysis in Engineering : Techniques, Tools, and Trends. CRC Press (Apr 2016). https://doi.org/10.1201/b21429
18. Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional synthesis of modular nonblocking supervisors. IEEE Transactions on Automatic Control **59**(1), 150–162 (Jan 2014)
19. 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 (Mar 2016). https://doi.org/10.1007/s10626-015-0217-y

20. Queiroz, M.H.d., 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, Springer US (2000)
21. 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 (Jan 1987)
22. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE **77**(1), 81–98 (Jan 1989)
23. 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 (Jan 2012). https://doi.org/10.1109/TSMCC.2011.2106495
24. 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: IEEE Conf. on Control Technology and Applications. pp. 458–464 (Aug 2018). https://doi.org/10.1109/CCTA.2018.8511449
25. 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 Conf. on Control Technology and Applications. pp. 1562–1568 (Aug 2017). https://doi.org/10.1109/CCTA.2017.8062679
26. Reijnen, F.F.H., Verbakel, J.J., van de Mortel-Fronczak, J.M., Rooda, J.E.: Hardware-in-the-loop set-up for supervisory controllers with an application: the Prinses Marijke complex. In: IEEE Conf. on Control Technology and Applications. p. accepted (Aug 2019)
27. Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., Rooda, J.E.: Supervisory control synthesis for a lock-bridge combination. Discrete Event Dynamic Systems (2019), submitted
28. Rudie, K., Wonham, W.M.: Think globally, act locally: decentralized supervisory control. IEEE Transactions on Automatic Control **37**(11), 1692–1708 (Nov 1992)
29. 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 (Jun 2009)
30. Su, R., van Schuppen, J.H., Rooda, J.E., Hofkamp, A.T.: Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. Automatica **46**(6), 968–978 (Jun 2010)
31. Theunissen, R.J.M.: Supervisory Control in Health Care Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven (2015), http://repository.tue.nl/786117
32. Wonham, W.M., Ramadge, P.J.G.: Modular supervisory control of discrete-event systems. Mathematics of Control, Signals and Systems **1**(1), 13–30 (Feb 1988)
33. Wonham, W.M., Cai, K.: Supervisory Control of Discrete-Event Systems. Springer, 1st edn. (2018)
34. Zhong, H., Wonham, W.M.: On the consistency of hierarchical supervision in discrete-event systems. IEEE Transactions on Automatic Control **35**(10), 1125–1134 (Oct 1990)