Structuring Multilevel Discrete-Event Systems with Dependency Structure Matrices

Martijn Goorden, Joanna van de Mortel-Fronczak, Michel Reniers, *Senior Member, IEEE*, Wan Fokkink, and Jacobus Rooda

***Abstract*—Despite the correct-by-construction property, one of the major drawbacks of supervisory control synthesis is state-space explosion. Several approaches have been proposed to overcome this computational difficulty, such as modular, hierarchical, decentralized, and multilevel supervisory control synthesis. Unfortunately, the modeler needs to provide additional information about the system's structure or controller's structure as input for most of these non-monolithic synthesis procedures. Multilevel synthesis assumes that the system is provided in a tree-structured format which may resemble a system decomposition. In this paper, we present a systematic approach to transform a set of plant models and a set of requirement models provided as extended finite automata into a tree-structured multilevel discrete-event system to which multilevel supervisory control synthesis can be applied. By analyzing the dependencies between the plants and the requirements using dependency structure matrix techniques, a multilevel clustering can be calculated. With the modeling framework of extended finite automata, plant models and requirements depend on each other when they share events or variables. We report on experimental results of applying the algorithm's implementation on several models available in the literature to assess the applicability of the proposed method. The benefit of multilevel synthesis based on the calculated clustering is significant for most large-scale systems.**

***Index Terms*—Supervisory control, multilevel discrete-event systems, dependency structure matrices, extended finite automata.**

I. Introduction

THE complexity of high-tech systems has increased over the last few decades due to increasing market demands for better performance and verified safety. Furthermore, the time-to-market has to be decreased while the quality of the engineering process has to be increased. Model-based systems engineering approaches provide support for dealing with these demands in the context of supervisory controller design. In this paper, discrete-event systems (DESs) are considered for which supervisory controllers need to be developed. The supervisory control theory (SCT) of Ramadge-Wonham [1], [2] provides an approach to synthesize supervisory controllers such that the controlled system behavior exhibits the specified behavior.

A DES can be modeled with Extended Finite Automata (EFAs), see [3]. Extended finite automata are finite automata enhanced with discrete variables. This allows for more compact model representation as shown in [4] and for the usage of state-based requirements as introduced in [5], [6]. Furthermore, EFAs allow for efficient symbolic computations with binary decision diagrams, see [7].

A major drawback of synthesizing supervisory controllers is the step where the supremal controllable language is calculated. 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 used to represent the system, as already observed in [2]. Several attempts exploiting different architectures are proposed to overcome these computational difficulties: modular [8]–[12], hierarchical [13]–[16], decentralized [17]–[20], distributed [21]–[25], coordinated [9], [26]–[28], compositional [29]–[31], and, more recently, multilevel supervisory control synthesis [32].

Although such techniques help in reducing the complexity of synthesis, applying them may require more effort from an engineer than monolithic synthesis. A problem with several of these supervisory control architectures is that, besides the plant models and control requirements, additional information about the system's structure or controller's structure needs to be provided as input for synthesis. For example, hierarchical supervisory control needs a hierarchical mapping of events or traces between the different levels, decentralized control requires projections to the subsystem alphabets, and multilevel control needs a tree-structured system. Sometimes, additional properties need to be satisfied, for example coobservability in decentralized control, see [18]. For those supervisory control synthesis procedures that require additional information, often a systematic approach is missing in the literature to transform any DES plant models together with control requirements to the appropriate input needed for such a procedure.

In this paper, we exploit the structure embedded in the set of plant models and the set of requirement models by using Dependency Structure Matrices (DSMs). A DSM is an $N \times N$ matrix capturing the dependencies among N system elements. A DSM provides a concise representation for the analysis of the structure of systems in many areas of engineering and research, see for industrial examples [33], [34]. With appropriate analysis techniques, such as clustering and sequencing, one is able to highlight important aspects in system structures, such as modules of system elements and cycles of process steps, respectively.

In addition to most suggestions found in the SCT literature to analyze the relationship between plant models (e.g., shared events in [35], [36]), we analyze the relationship within the combined set of plant models and the requirement models, as also suggested in [37]. The motivation is twofold: (1)

M. Goorden, J. van de Mortel-Fronczak, M. Reniers, W. Fokkink and J. Rooda are with Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. (email: m.a.goorden@tue.nl, j.m.v.d.mortel@tue.nl, m.a.reniers@tue.nl, w.j.fokkink@tue.nl, j.e.rooda@tue.nl)

This work is supported by Rijkswaterstaat, part of the Ministry of Infrastructure and Water Management of the Government of the Netherlands.

Manuscript received ; revised

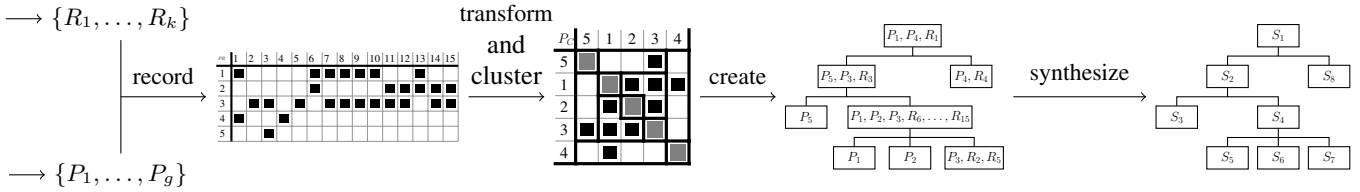

Fig. 1. Overview of the proposed method. It starts with a given set of plant models $\{P_1, \ldots, P_g\}$ and a given set of requirement models $\{R_1, \ldots, R_k\}$. First, the dependencies between the plant models and the requirement models are *recorded* in a rectangular Domain Mapping Matrix (DMM). Then, this DMM is *transformed* into a Dependency Structure Matrix (DSM) and subsequently *clustered*. From the clustered DSM, a Multilevel Discrete Event System (MLDES) is *created*. Finally, multilevel synthesis of [32] can be applied to *synthesize* a supervisor for each node in the MLDES.

supervisory control synthesis requires both plant models and requirement models, and (2) plant models of realistic systems are often not directly related to each other and only indirectly via the requirements, see for example [38]. The work of [37] introduces control-flow nets to analyze dependencies in the system and subsequently abstract away those parts of the system that will not contribute to a potential blocking issue. Control-flow nets are defined for shuffle systems with server and buffer specifications, which limits the applicability. The introduction of DSMs could be seen as a generalization of control-flow nets.

The contribution of this paper is a proposal for a systematic approach to transform a set of plant models and a set of requirement models into a tree-structured multilevel discrete-event system with the properties needed for multilevel supervisory control synthesis of [32]. Multilevel synthesis is chosen as it resembles the multilevel (or decompositional) way of thinking of engineers and directly raises the question how to obtain a multilevel structure. The proposed method is summarized in Figure 1. First a domain mapping matrix is created that relates plant models to requirement models. A DSM is constructed from this domain mapping matrix with the plant models as system elements. Two plant models are related to each other when there exists a requirement that depends on both plant models. The DSM is clustered to identify a clustering of plant models. Finally, this clustering is transformed into a tree-structured multilevel discrete-event system (MLDES) such that multilevel supervisory control synthesis can be applied resulting in a supervisor for each node.

This paper is an extended version of [39]. Firstly, this paper uses the modeling formalism of EFAs, instead of finite automata (FAs). By using EFAs, plant models and requirement models depend on each other by shared events and shared variables, instead of just shared events in the FA formalism. Therefore, recording the dependencies should be changed accordingly. The two subsequent steps in the proposed method turn out to be independent of the chosen modeling formalism. In the last step, when supervisors are synthesized for each node, an EFA-based synthesis algorithm should be used instead of an FA-based one. Furthermore, defining the proposed method for EFAs makes it applicable to both EFAs and FAs, since each FA is also an EFA. The second extension is the inclusion of numerical results on the effect of clustering parameters on the computational complexity reduction of supervisor synthesis, on the effect of different clusterings on nonblocking control, and on the applicability of the proposed method on small and large benchmark models.

P	1	2	3	4	5
1	-			1	
2		-	1	1	1
3		1	-		1
4	1	1		-	
5		1	1		-

P_C	1	4	2	5	3
1	-	1			
4	1	-	1		
2		1	-	1	1
5			1	-	1
3			1	1	-

Fig. 2. Left the unclustered example DSM P and right the clustered DSM P_C revealing two clusters.

This paper is structured as follows. The concepts and notations used are provided in Section II regarding DSMs and in Section III regarding SCT. The main results are presented in Section IV with an illustrative example of a DES model of a lock. Section V provides results of three experiments with the implementation of the proposed method. The conclusions are presented in Section VI.

II. Dependency Structure Matrices

In this section, the concepts and notations related to *Dependency Structure Matrices* (also called Design Structure Matrices) used in this paper are summarized. A DSM is a square matrix with the same entities along its axis (e.g., components of a system) and cells representing relationships between the entities (e.g., a spatial relationship). These relationships can be different per DSM. DSMs with a single kind of off-diagonal mark are called binary DSMs, while DSMs with off-diagonal cells containing numbers are called numerical DSMs. Fig. 2 left shows an example DSM P of a system with five entities numbered 1 through 5. A relationship between two entities is indicated with a 1. The absence of a relation is indicated with an empty matrix entry. Therefore, this DMS is a binary DSM.

A matrix in which the relationships between different domains are described is called a *Domain Mapping Matrix* (DMM), which is a rectangular matrix. A DMM can also be binary or numerical. The generation of DSMs from DMMs with matrix multiplications is described in [40], [41].

There exist different types of DSMs. Undirected relationships result in a static DSM, while directed relationships result in a dynamic DSM. The different types of DSMs allow for different types of analyses of the considered system. In this

paper, a static DSM is analyzed. Often, the goal of analyzing static DSMs is to find a modular structure by clustering the entities of the DSM, as shown, for example, in [42]. Fig. 2 right shows the clustered example DSM P_C. By reordering the rows and columns of the unclustered DSM P, strongly related entities are placed together to form a cluster. Entities 1 and 4 form a cluster and entities 2, 5, and 3 form a cluster. These clusters are called modular clusters as each entity of the system is included in exactly one of the clusters. Several heuristics exists in literature to cluster a DSM, see for example [42]. Typically, different settings for these heuristics will result in different clusters for the same DSM.

In [33], a more in-depth introduction to DSM analysis, including notions not used in this paper, is given. Examples and applications of DSMs can be found in the recent review paper [34].

A. Multilevel Markov clustering

The clustering algorithm of [42] utilizes Markov clustering [43]. In Markov clustering, a symmetric stochastic matrix P is used that represents the transition matrix of a Markov chain. The clustering algorithm is an iterative process where each iteration k consists of two steps: the expansion step and the inflation step.

In the expansion step, the transition matrix of the previous step P_{k-1} is raised to the power α to obtain $P_k = P_{k-1}^{\alpha}$. The new transition matrix represents the transition probabilities of a Markov chain where a random walker has taken α steps. In the inflation step, high transition probabilities are increased and low transition probabilities are decreased by taking the Hadamard (entry wise) power of P_k with coefficient β and then normalizing the columns.

The Markov clustering terminates when a fixed-point is reached [43]. The resulting invariant matrix is then interpreted as the adjacency matrix of a weighted directed graph denoting disjoint clusters.

To apply Markov clustering on a DSM, the DSM has to be converted into a transition matrix of a Markov chain. To this end, the DSM is interpreted as an adjacency matrix of a weighted directed graph, where the rows and columns are the nodes and the entries are the weights. For each node, a positive fluid is injected to determine the influence of this node on other nodes, while a negative fluid is injected to determine the dependency of this node on other nodes. The strength of the influence and dependency decreases with a factor μ each time the flow passes through a node.

This Markov clustering is turned into a multilevel Markov clustering by using graph coarsening. All elements within a cluster are collapsed into a new super-node. The original DSM is coarsened with these new super-nodes, where the new weight between the super-nodes equals the sum of the inter-cluster edge weights between the clusters.

III. Multilevel Discrete-Event Systems

In this section, the concepts and notations of Supervisory Control Theory used in this paper are summarized. A more in-depth introduction to SCT, including notions not used in this paper, can be found in [44], [45].

A. Preliminaries

Discrete-event systems can be modeled with extended finite automata (EFAs). An EFA using variable set V with initial valuation $\hat{v}_0$ is a 5-tuple $(L, \Sigma, \rightarrow, L_m, l_0)$ where L is a finite set of locations, Σ an alphabet, $\rightarrow \subseteq L \times \Sigma \times \Pi_V \times L$ the transition relation with Π_V the set of all update functions using variables from V, $L_m \subseteq L$ a set of marked locations indicating 'accepting' or 'final' locations, and $l_0 \in L$ the initial location.

The alphabet Σ is partitioned into two disjoint sets: the set of controllable events Σ_c and the set of uncontrollable events Σ_{uc}. Controllable events can be disabled by the supervisor, e.g., switching on an actuator, while uncontrollable events cannot be disabled, e.g., switching sensor values. We introduce the function $Alp(E)$ to represent the alphabet of EFA E.

In an EFA, each transition is augmented with an update using variables, constants, the Boolean literals true (**T**) and false (**F**), and the usual arithmetical and logical connectives, see [46]. With each variable $v \in V$, a domain $\mathrm{dom}(v)$ of values is associated. A valuation is a mapping $\hat{v} : V \rightarrow \bigcup_{v \in V} \mathrm{dom}(v)$ with $\hat{v}(v) \in \mathrm{dom}(v)$ for each $v \in V$. The set of all valuations on V is denoted by $\mathrm{Val}(V)$. The initial valuation is denoted by $\hat{v}_0$. An example of an update function is $p(\hat{v}_1, \hat{v}_2) \equiv \hat{v}_1(v_1) = 1 \wedge \hat{v}_2(v_1) = 2$ where $\hat{v}_1$ and $\hat{v}_2$ denote the current-state and next-state valuations, respectively. This update function evaluates to true if the current-state value of v_1 equals 1 and the next-state value equals 2. Formally, an update function is a predicate function $u : \mathrm{Val}(V) \times \mathrm{Val}(V) \rightarrow \mathbb{B}$. The set of all update functions using variables from V is denoted by Π_V. A more detailed discussion on predicates and combining predicates can be found in [47]. From now on we assume that there is a globally defined variable set V together with domains of each variable and initial valuation $\hat{v}_0$.

In an EFA, a transition $(l_1, \sigma, u, l_2) \in \rightarrow$ can be taken if $u(\hat{v}_1, \hat{v}_2)$ evaluates to true with the current-state valuation $\hat{v}_1$ and next-state valuation $\hat{v}_2$. After taking the transition, the current location l_1 of the EFA has been updated to l_2 and the valuation $\hat{v}_1$ to $\hat{v}_2$.

It is not necessary to use all variables in an update. Let $Var(u) \subseteq V$ denote the variables used in update u. This function can be extended to an EFA as $Var(E) = \bigcup_{(l_1,\sigma,u,l_2) \in \rightarrow} Var(u)$ providing the set of variables used somewhere in this EFA.

An automaton is called prefix-closed if all locations are marked. In a prefix-closed automaton it holds that for each path from the initial location to a marked location all prefixes also lead to a marked state.

For large-scale systems, the model is given as a collection of EFA models $G_s = \{G_1, G_2, \ldots, G_m\}$. Such a collection is called a *composed system*. Two EFAs can be combined by using the synchronous composition, see [46].

Definition 1 (Synchronous composition): Let $G^k = (L^k, \Sigma^k, \rightarrow^k, L_m^k, l_0^k), k = 1, 2$ be EFAs with variable set V. The synchronous composition of G^1 and G^2 is

$$G^1 \parallel G^2 = (L^1 \times L^2, \Sigma^1 \cup \Sigma^2, \rightarrow, L_m^1 \times L_m^2, (l_0^1, l_0^2))$$

where the transition relation $\rightarrow$ is defined as

- $((l_1^1, l_1^2), \sigma, u, (l_2^1, l_2^2)) \in \rightarrow$ if $\sigma \in \Sigma^1 \cap \Sigma^2$ and there exist $(l_1^1, \sigma, u^1, l_2^1) \in \rightarrow^1$ and $(l_1^2, \sigma, u^2, l_2^2) \in \rightarrow^2$ such that $u = u^1 \wedge u^2$;
- $((l_1^1, l_1^2), \sigma, u^1, (l_2^1, l_1^2)) \in \rightarrow$ if $\sigma \in \Sigma_1 \setminus \Sigma_2$ and $(l_1^1, \sigma, u^1, l_2^1) \in \rightarrow^1$;
- $((l_1^1, l_1^2), \sigma, u^2, (l_1^1, l_2^2)) \in \rightarrow$ if $\sigma \in \Sigma_2 \setminus \Sigma_1$ and $(l_1^2, \sigma, u^2, l_2^2) \in \rightarrow^2$.

Synchronous composition is associative, i.e., $(G_1 \parallel G_2) \parallel G_3 = G_1 \parallel (G_2 \parallel G_3) = G_1 \parallel G_2 \parallel G_3$.

A less frequently used notion is that of a product system representation of FAs [2], which is here adapted for EFAs. Let $\mathcal{P}(G_s)$ be the set of all partitions of the composed system G_s, which is a lattice [48]. In the remainder, we write $\mathcal{P}$ instead of $\mathcal{P}(G_s)$ for notational simplicity. A partition $P \in \mathcal{P}$ is called a *product system* if for every pair of cells $p_1, p_2 \in P$, if $p_1 \neq p_2$ then the alphabets are disjoint, i.e., $Alp(p_1) \cap Alp(p_2) = \emptyset$ and $Var(p_1) \cap Var(p_2) = \emptyset$ where $Alp(p) = \bigcup_{G_i \in p} Alp(G_i)$ and $Var(p) = \bigcup_{G_i \in p} Var(G_i)$. The infimum of the set of all product systems PS is called the *most refined product system partitioning* and the *most refined product system* $G'_s = \{\|_{G_i \in p} G_i \mid p \in \inf(PS)\}$.

Proposition 1 (Most refined product system): Let G_s be a composed system. Then $\inf(PS) \in PS$.

Proof: As the lattice of partitions is finite, and the meet of two product systems is again a product system, it follows that $\inf(PS) \in PS$. ∎

EFA models in a product system representation do not interact with each other, i.e., they behave asynchronously. The product system representations ease the reasoning about the system, see [10], [37]. Any composed system G_s can be transformed into a product system representation; the trivial one is just the synchronous product of all models. The most-refined product system can be seen as the one obtained with the least number of synchronous product operations.

Finally, in SCT a distinction is made between plant models and requirement models. Plant models describe the uncontrolled behavior of a system, while requirement models describe the desired behavior of the system.

Requirements can be expressed with both EFAs and state-based expressions [5], [6]. *Mutual state exclusion* requirements refer to forbidden combinations of states of the plant models in the composed system. We define the alphabet $Alp(K)$ of a mutual state exclusion expression K as the empty set and the set of variables $Var(K)$ as all variables used in K.

B. Monolithic supervisor synthesis

The objective is to design a controller whose function is to disable controllable events so that the closed loop system of the plant and the controller obeys the specified behavior. SCT provides a method to synthesize a supervisor that adheres to the following control objectives for given plant and requirement models, see [49].

- *Safety*: all possible behavior of the controlled system should always satisfy the imposed requirements.
- *Controllability*: uncontrollable events may never be disabled by the supervisor.
- *Non-blockingness*: the controlled system should be able to reach a marked state from every reachable state.
- *Maximal permissiveness*: the supervisor does not restrict more behavior than strictly necessary to enforce safety, controllability, and non-blockingness.

Monolithic supervisory control synthesis results in a single supervisor S from a single plant model and a single requirement model. We refer to S as the automaton representation of the synthesized supervisor. When the plant model and the requirement model are given as a collection of models P_s and R_s, respectively, the monolithic plant model P and requirement model R are obtained by performing the synchronous products of the models in the respective collection.

C. Multilevel discrete-event systems

A *multilevel discrete-event system* (MLDES) is a system with a tree-based structure, as recently proposed in [32]. More formally, let T represent an index set for a tree structure, where each element $n \in T$ represents a node in the tree. An MLDES is defined as a set G_s of subplants, $\{G_n \mid n \in T\}$, such that global plant G is given by $G = \|_{n \in T} G_n$.

The control synthesis problem of MLDES is stated as follows. Consider the supervisory control problem where the global plant is given as $G = \|_{n \in T} G_n$ and the global requirement $K = \|_{n \in T} K_n$. Find a set of supervisors where the global supervisor is given by $S = \|_{n \in T} S_n$ such that the controlled system $S \parallel G$ satisfies safety, controllability, nonblockingness, and maximal permissiveness.

As shown in [32], the set of supervisors S_s can be constructed by synthesizing for each node $n \in T$ a supervisor S_n with monolithic supervisory control synthesis. The following theorem states that there exists a solution satisfying safety for FA models.

Theorem 1 (Existence of MLDES supervisors [32]): Consider an MLDES with FA-based subplant set G_s and a set of prefix-closed, controllable requirements $\{K_n \mid n \in T \wedge K_n \subseteq \Sigma^*_{G_n}\}$ such that $K = \|_{n \in T} K_n$. There exists a set of supervisors S_s, where $S_s = \{S_n \mid n \in T \wedge S_n \parallel G_n = K_n\}$, such that $S \parallel G = K$ with $S \parallel G = \|_{n \in T} S_n \parallel G_n$.

For prefix-closed requirements it can then also be shown that the solution is maximally permissive and nonblocking [32], [44]. Otherwise, a nonblocking check, for example the one described in [35], should be performed to assess whether the system controlled by the resulting set of supervisors is nonblocking. As stated by [32], the case of non-prefix closed requirements in conjunction with controllability is part of future research.

In [32], only FA-based models are considered. The extension of Theorem 1 to EFA-based models is part of ongoing research.

IV. Proposed method

In this section, the proposed method of transforming a set of plant models and requirement models into an MLDES is described. The input for the method can be any set of plant models and requirement models. The transformation consists of three stages: recording the dependencies between

Fig. 3. The lock at Terneuzen, The Netherlands. Image from https://beeldbank.rws.nl, Rijkswaterstaat.

the models, finding a valid clustering of the composed system, and constructing the MLDES. These three stages are explained in detail below. Finally, the complete algorithm is provided.

In the remainder of this paper, a problem definition denotes any composed system, i.e., $G = \|_{i \in I} G_i,\ I = \{1, 2, \ldots, g\},\ g \in \mathbb{N}^+$, together with the composed global requirement, i.e., $K = \|_{j \in J} K_j,\ J = \{1, 2, \ldots, k\},\ k \in \mathbb{N}^+$.

Example *Throughout this section, an illustrative example of a lock is provided to explain the presented method. To maintain different water levels within a canal, a lock is constructed which allows ships to be lifted to the higher water level or to be lowered to the lower level. Fig. 3 shows the lock located at Terneuzen, The Netherlands.*

The following subplants are present in this simplified system:

- *Side 1 entering light*
- *Side 1 leaving light*
- *Side 1 gate actuator*
- *Side 1 gate sensor*
- *Side 1 sewer actuator*
- *Side 1 sewer sensor*
- *Side 1 equal-water sensor*
- *Side 2 entering light*
- *Side 2 leaving light*
- *Side 2 gate actuator*
- *Side 2 gate sensor*
- *Side 2 sewer actuator*
- *Side 2 sewer sensor*
- *Side 2 equal-water sensor*

On this system, 30 requirements are imposed to guarantee the safe operation of the lock. These requirement are formulated by engineers of Rijkswaterstaat. An example of a safety requirement is that if there is no equal water level over a gate, then the gate may not be opened. *The complete model, and all other models used later in this paper, can be found in [50].*

A. Recording the dependencies

The relationships within the problem definition are analyzed. Since plant models and requirement models have a different role in the synthesis process, we consider them as different domains. The dependencies between plants and requirements result in a DMM. From this domain mapping, we could create in a later stage a DSM with plants as entities and a DSM with requirements as entities, both using simple matrix multiplications, see [40].

PR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1						1	1	1	1	1			1																	
2						1					1	1	1	1	1															
3		1	1		1		1	1	1	1	1	1		1	1				1	1									
4	1			1												1	1													
5			1																											
6																					1	1	1	1	1			1		
7																					1					1	1	1	1	1
8				1	1												1	1		1		1	1	1	1	1	1		1	1
9	1	1														1			1											
10																		1												

Fig. 4. The DMM PR of the simple lock. Only the nonzero elements are shown for readability.

As a first step, we transform the general problem definition into the most refined product system. Therefore, if we later refer to an event or a variable, there will be only a single plant model containing this event in the alphabet or using this variable. Let the most refined product system be denoted by $\{G'_i \mid i \in I'\}$ with $G = \|_{i \in I'} G'_i,\ I' = \{1, 2, \ldots, g'\},\ g' \leq g$.

Let the DMM be denoted by PR. Construct PR such that $PR(i, j) = 1$ if the alphabets or the used variables sets of component i and requirement j are not disjoint, else $PR(i, j) = 0$. During the supervisory control synthesis procedure, it may be possible that the behavior of G'_i is restricted by requirement K_j if $PR(i, j) = 1$. If $PR(i, j) = 0$, we know that requirement K_j has no effect on the behavior of plant model G'_i during the supervisory control synthesis procedure. Therefore, a binary DMM is sufficient and no need for a numerical DMM is present.

Example *The most refined product system of the simplified lock is given by:*

1) Side 1 entering light
2) Side 1 leaving light
3) Side 1 gate
4) Side 1 sewer
5) Side 1 equal-water sensor
6) Side 2 entering light
7) Side 2 leaving light
8) Side 2 gate
9) Side 2 sewer
10) Side 2 equal-water sensor

Instead of 14 components, the most refined product system has 10 components. On each side, the gate actuator and sensor have been merged, as well as the circulation sewer actuator and sensor. The numbers before the components are used in the remainder of this section to refer to a particular component of the product system.

The resulting DMM of the simple lock example is shown in Fig. 4. Consider the requirement mentioned before: if there is no equal water over a gate, then the gate may not be opened. *For side 1, this is the third requirement. This requirement has a relationship with the gate component (number 3) and the equal water sensor component (number 5). Therefore,* $PR(3, 3) = 1$, $PR(5, 3) = 1$, *and all other elements in column 3 of* PR *are zero. This is done for all 30 requirements.*

B. Finding a valid clustering

Before we can find clusters, we need to define a multilevel clustering. A multilevel clustering can be seen as recursively

P	1	2	3	4	5	6	7	8	9	10
1	6	2	4							
2	2	6	4							
3	4	4	13		1			2	2	
4				4				2	2	
5			1		1					
6						6	2	4		
7						2	6	4		
8			2	2		4	4	13		1
9			2	2					4	
10								1		1

P_C	1	2	3	5	6	7	8	10	4	9
1	6	2	4							
2	2	6	4							
3	4	4	13	1			2			2
5			1	1						
6					6	2	4			
7					2	6	4			
8			2		4	4	13	1	2	
10							1	1		
4							2		4	2
9			2						2	4

Fig. 5. DSM P for the simple lock example: left the unclusterd P and right the clustered P_C.

partitioning set A, i.e., set A is partitioned into $\{A_1, \ldots, A_s\}$ where each cell A_i is again partitioned, and so on until partitions with a single element are reached.

Definition 2 (Multilevel clustering): The set of all multilevel clusterings C_A^m on a non-empty element set A is inductively defined.

- When $|A| = 1$, $(A, A) \in C_A^m$.
- $\{A_1, \ldots, A_s\}$ is a partition of A for some $s \geq 2$, and $\forall 1 \leq i \leq s : (A_i, M_i) \in C_{A_i}^m$ with M_i the multilevel clustering of A_i, then $(A, \{(A_i, M_i) \mid 1 \leq i \leq s\}) \in C_A^m$.

In tuple $(A, M) \in C_A^m$, A provides immediately all elements in this multilevel clustering and set M contains the multilevel clusterings of its children. For example, $(\{1,2,3\}, \{(\{1\},\{1\}),\ (\{2,3\}, \{(\{2\},\{2\}),(\{3\},\{3\})\})\})$ is a multilevel clustering on the set $\{1,2,3\}$.

A multilevel clustering will function as a basis for creating the tree index set T in the next step. To find a multilevel clustering of the product system $G = \|_{i \in I'} G'_i$, we first transform DMM PR into a DSM with the plants as the domain. Let DSM P be defined as $P = PR \cdot PR^T$ with PR^T the transpose matrix of PR.

By creating DSM P from DMM PR, we have the following interpretation. When $P(a, b) = k$ we know that there exist k requirements that use events or variables from both G'_a and G'_b to describe the desired behavior. Therefore, in a multilevel clustering based on DSM P a cluster of plant models indicates that multiple requirements relate the plant models from the cluster. Plant models from different clusters are related by none or only some requirements.

Example *Fig. 5 left shows the DSM P of the lock example. For example, cell $P(3,1) = 4$ indicates that there are four requirements which use both plant component 1 (entering light side 1) and plant component 3 (gate side 1). Furthermore, the elements on the diagonal indicate the number of requirements related to that particular plant component.*

The clustering algorithm as presented in [42] is used to find a multilevel clustering. This paper does not discuss which clustering algorithm is the best for our purpose of eventually synthesizing supervisors. Any valid multilevel clustering according to Definition 2 can be used. Due to the bottom-up partitioning approach used in [42], a valid multilevel clustering is calculated. Furthermore, it does not require any information on the structure of the multilevel clustering as input, like the number of expected clusters or the number of hierarchical layers.

Example *Fig. 5 right shows the clustered DSM P_C for the simple lock model. The clustering is generated with the algorithm presented in [42] with the parameter values $\alpha = 2, \beta = 2.2, \gamma = 10$ and $\mu = 2.0$. There are three clusters indicated: a cluster with the entering light, leaving light, gate, and equal water sensor of side 1, a similar cluster with elements of side 2, and a cluster with the circulation sewers of both sides.*

C. Constructing the MLDES

From now on we assume that we have a multilevel clustering (I', M) on index set I' of the most refined product system $\{G'_i \mid i \in I'\}$. We will use the information from DSM P and DMM PR to construct the index set T of the tree structure together with the set of plant models $\{G_n \mid n \in T\}$ and the set of requirement models $\{K_n \mid n \in T\}$.

In this section, we present algorithms to construct the tree index set T and the problem definition at each node in the tree. We first explain the algorithm to construct the index set of the tree structure. Then we explain the algorithm to determine the problem definition.

Algorithm 1 Transform$C^m$to$T$

Input: multilevel clustering (A, M), new node index n, DSM P of (A, M), DMM PR, most refined product system $\{G'_i\}$, set of requirements $\{K_j\}$

Output: index set of tree-structure T of (A, M), set of plant models $G_T = \{G_n \mid n \in T\}$, set of requirement models $K_T = \{K_n \mid n \in T\}$, last used node index m

```
Set T = {n} and m = n
G_n, K_n, P_mod, PR_mod =
    CalculateG_n and K_n((A, M), P, PR, {G'_i}, {K_j})
Set G_T = {G_n}, K_T = {K_n}
if size(M) > 1 then
    for all (A_p, M_p) ∈ M do
        T_p, G_Tp, K_Tp, m_p =
            TransformC^m toT((A_p, M_p), m + 1,
                P_mod, PR_mod, {G'_i}, {K_j})
        T = T ∪ T_p, G_T = G_T ∪ G_Tp, K_T = K_T ∪ K_Tp
        m = m_p
    end for
end if
```

Algorithm 1 creates the index set T of the tree structure embedded in the given multilevel clustering (A, M). We apply this algorithm recursively to do a depth-first search through the multilevel clustering (I', M) obtained from the previous step. This algorithm consists of two parts: first, the plant model and requirement model are calculated for the new node with Algorithm 2, and second, we explore new nodes further down the tree and repeat the algorithm. Initially, Algorithm 1 is called with Transform$C^m$to$T$ $((I', M), 1, P, PR, \{G'_i | i \in I'\}, \{K_j | j \in J\})$.

When Algorithm 1 is performed, we know that there exists a valid level in the tree structure. This level n is added to

the tree structure in Line 1 and noted that this is the last tree index currently used. Furthermore, in Lines 2-3 the plant model and requirement model on level n are calculated, see Algorithm 2. This algorithm modifies the DSM P and DMM PR. The algorithm continues if we can go further down in the tree structure. At Lines 4-10 we go further down the tree structure for each subcluster. In Line 6 the recursive call is performed. The results of this call are added to the already obtained results in Lines 7-8. The order of the calls chosen in Line 5 only influences the order of the nodes.

The following proposition ensures that Algorithm 1 terminates.

Proposition 2 (Termination of Algorithm 1): Given a valid multilevel clustering (A, M) with A a finite set, Algorithm 1 terminates.

Proof: A new inductive call of Algorithm 1 is performed when the cardinality of the current multilevel clustering (A, M) is larger than 1. Furthermore, observe that the new inductive calls are performed on subclusters of the current multilevel clustering. For each subcluster (A_p, M_p) of (A, M) it holds that $|A_p| < |A|$ and thus finite. Therefore, we can construct the following inductive proof.

Inductive base When $|A| = 1$, Algorithm 1 terminates.

Inductive hypothesis Assume that $\forall (A_p, M_p) \in M$, Algorithm 1 will terminate eventually for (A_p, M_p).

Inductive step In Algorithm 1 inductive calls are performed for all $(A_p, M_p) \in M$. Since we know by the inductive hypothesis that each of these inductive calls will terminate, all inductive calls in Algorithm 1 will terminate. Therefore, the call of Algorithm 1 with (A, M) will terminate. ■

Example *Starting from the root node, we can identify three subclusters:* $\{1, 2, 3, 5\}, \{6, 7, 8, 10\}$*, and* $\{4, 9\}$*. Searching further, we see that the first subcluster has four child nodes, the second subcluster also has four child nodes, and the third subcluster has two child nodes. The resulting tree* T *has three levels and is shown in Fig. 6.*

Algorithm 2 shows the procedure to calculate the problem definition for a certain node n in the tree structure. Conceptually, the algorithm uses the DMM PR and DSM P to calculate the plant model and requirement model for the node as follows. If a singleton has been reached in the multilevel clustering, a leaf node in the tree structure is reached, the plant model in this singleton is directly placed in the node, and requirements are identified that only have a dependency with this plant model (and no other plant model). In all other cases of multilevel clusterings, a regular node in the tree has been reached, requirement models for this node are identified based on dependencies outside the subclusters and within the current cluster, and the plant models are those that relate to the identified requirement models.

In detail, Algorithm 2 works as follows, where $P(:, j)$ indicates column vector j of matrix P. In Line 1 we create P_{mod} and PR_{mod} as we may modify it. We need to make a distinction between a leaf node and a non-leaf node.

If we have reached a leaf node, then in Lines 3-4, plant index set I_n at this node is set to the plant of the product system indicated by the multilevel clustering. For this single subplant, there may exist requirements which are only related to this subplant. These requirements are identified in Line 4.

Algorithm 2 Calculate$G_n$andK_n

Input: multilevel clustering (A, M), DSM P of (A, M), DMM PR, most refined product system $\{G'_i\}$, set of requirements $\{K_j\}$

Output: Plant model G_n for top cluster of (A, M), requirement model K_n of top cluster (A, M), modified DSM P_{mod}, modified DMM PR_{mod}

1: $P_{mod} = P, PR_{mod} = PR$
2: **if** size$(M) = 1$ **then**
3: $\quad I_n = A$
4: $\quad J_n = \{j \in J \mid PR(A, j) = 1 \wedge \sum_{i \in I'} PR(i, j) = 1\}$
5: **else**
6: $\quad$ **for all** $(A_x, M_x), (A_y, M_y) \in M, A_x \neq A_y$ **do**
7: $\quad\quad$ **for all** $x \in A_x, y \in A_y$ **do**
8: $\quad\quad\quad$ **if** $P_{mod}(x, y) \neq 0$ **then**
9: $\quad\quad\quad\quad J_{temp} = \{j \in J \mid PR_{mod}(a, j) = 1 \wedge PR_{mod}(b, j) = 1\}$
10: $\quad\quad\quad\quad P_{mod} = P_{mod} - \sum_{j \in J_{temp}} PR_{mod}(:, j) \cdot PR_{mod}(:, j)^T$
11: $\quad\quad\quad\quad I_n = I_n \cup \{i \in I' \mid \exists j \in J_{temp} : PR_{mod}(i, j) = 1\}$
12: $\quad\quad\quad\quad J_n = J_n \cup J_{temp}$
13: $\quad\quad\quad\quad PR_{mod}(:, J_{temp}) = 0$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end for**
16: $\quad$ **end for**
17: **end if**
18: $G_n = \|_{i \in I_n} G'_i$
19: $K_n = \|_{j \in J_n} K_j$

If we did not reach a leaf node, the multilevel clustering (A, M) consists of multiple subclusters. At the current node, we need to identify the requirements which combine the subclusters into this particular multilevel clustering (A, M). To this end, we search in DSM P nonzero elements outside each of the subclusters, but inside cluster (A, M). In Lines 6-8 we consider all possible combinations of elements from two different subclusters. If we find a nonzero element in P_{mod}, we know that there exists at least one requirement which relates the two different subclusters to each other. In Line 9 we identify these requirements by searching DMM PR_{mod}. It is possible that one of these particular requirements also relates to other subclusters or even relates to elements inside a particular subcluster. To prevent placing a requirement in multiple nodes, we update P_{mod} and PR_{mod} by removing all relationships resulting from the found requirements. When we have found all requirements relating the subclusters together at this node, we calculate at Lines 18 and 19 the plant model and requirement model for this node.

Example *Consider the first node* 1*. In Fig. 5 we can identify three non-zero elements in* P *outside the clusters* $\{1, 2, 3, 5\}, \{6, 7, 8, 10\}$*, and* $\{4, 9\}$ *and in the lower triangular part:* $P(8, 3), P(4, 8)$ *and* $P(9, 3)$*. For the first element*

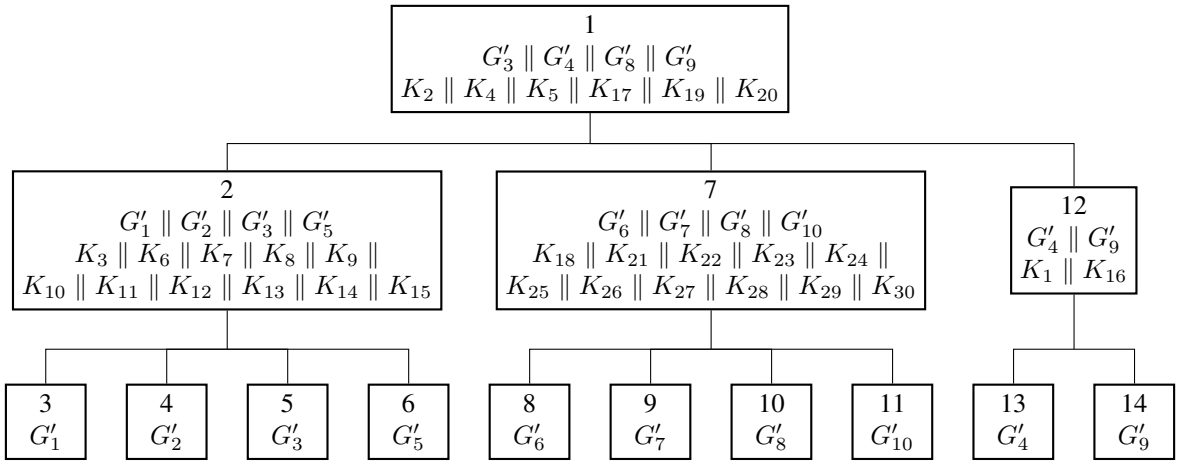

Fig. 6. Tree structure: index set T together with G_n and K_n.

$P(8,3)$, we search in PR for all $j \in J$ such that $PR(8,j) \neq 0 \wedge PR(3,j) \neq 0$. From Fig. 4, we can see that only $j = 5$ and $j = 20$ satisfy this condition and are therefore added to J_n. Repeating this for $P(4,8)$ and $P(9,3)$ results finally in $J_{(1,1,1)} = \{2,4,5,17,19,20\}$.

To create I_n, we search again PR to find those $i \in I'$ such that $PR(i,j) \neq 0, j \in J_n$. From Fig. 4 we can conclude that $I_{(1,1,1)} = \{3,4,8,9\}$.

The same approach can be applied to find G_n and K_n for each node $n \in T$. Fig. 6 shows the resulting plant and requirement models at each node. If a node does not show a requirement model K_n, none of the original requirements is placed at this node. A node will not get assigned any requirement if there are no requirements related to only the plant models in that node. For the simple lock model, there are no requirements related to only a single plant model, see the DMM in Fig. 4.

We can prove the following two Propositions 3 and 4, which state that every original plant model and requirement model is located somewhere in the constructed MLDES, respectively. The necessary and sufficient condition in Proposition 4 simply states that each requirement has a dependency with some plant model, which is a reasonable assumption for models of real systems.

Proposition 3 (Plant model conservation): Consider the MLDES constructed with Algorithm 1, it holds that $\|_{i \in I'} G_i = G = \|_{n \in T} G_n$.

Proof: For each node n visited by Algorithm 1, Algorithm 2 is called at Line 2. It follows from Algorithm 2, Line 18, that $G_n = \|_{i \in I_n} G'_i$ where $I_n \subseteq I'$. Thus,

$$\|_{n \in T} G_n = \|_{n \in T} (\|_{i \in I_n} G'_i) = \|_{i \in (\cup_{n \in T} I_n)} G'_i \subseteq \|_{i \in I'} G'_i. \quad (1)$$

For a valid multilevel clustering (I', M) on index set I' it holds that each index is included in exactly one leaf node. For the leaf node of element i, we know by Algorithm 2 that $G_n = G'_i$. Therefore, the subset equality in Equation 1 becomes a set equality, since $\cup_{n \in T, n}$ is leaf node $I_n = I'$. ∎

Proposition 4 (Requirement model conservation): Consider the MLDES constructed with Algorithm 1, it holds that $\|_{j \in J} K_j = K = \|_{n \in T} K_n$ if and only if $\forall j \in J : PR(:,j)^T \cdot PR(:,j) \geq 1$.

Proof: ($\Leftarrow$) For each node n visited by Algorithm 1, Algorithm 2 is called at Line 2. It follows from Line 19 of Algorithm 2 that $K_n = \|_{j \in J_n} K_j$. So, to prove that $K = \|_{n \in T} K_n$, it suffices to prove that $\cup_{n \in T} J_n = J$.

Requirement j is added to J_n for some n if one of the nonzero elements of $PR(:,j) \cdot PR(:,j)^T$ is encountered by Algorithm 2 when Algorithm 1 is at node n. From the assumption that $PR(:,j)^T \cdot PR(:,j) \geq 1$ it follows that there exists at least one nonzero element in $PR(:,j) \cdot PR(:,j)^T$. Therefore, we know that one of these nonzero elements is checked if we check all elements of $P = PR \cdot PR^T$.

All diagonal elements of P are checked at the leaf nodes of the tree structure. For all off-diagonal elements $P(a,b)$ with $a, b \in I'$ it holds that we always can find two multilevel clusters (A_x, M_x) and (A_y, M_y) such that $A_x \neq A_y \wedge a \in A_x \wedge b \in A_y \wedge \exists (A_p, M_p) : (A_x, M_x), (A_y, M_y) \in M_p$. Since Algorithm 1 starts at the root node of the tree structure and inductively creates the index set T until it has reached all leaf nodes, we must have found (A_p, M_p) such that Algorithm 2 checks $P(a,b)$. Therefore, all elements of P are checked, is one of the nonzero elements of $PR(:,j) \cdot PR(:,j)^T$ encountered, is j added to J_n for some n, and finally $\cup_{n \in T} J_n = J$.

($\Rightarrow$) From $\|_{j \in J} K_j = K = \|_{n \in T} K_n$ it follows that $\forall j \in J, \exists n \in T : j \in J_n$. Requirement j is added to J_n if one of the nonzero elements of $PR(:,j) \cdot PR(:,j)^T$ is encountered by Algorithm 2 when Algorithm 1 is at node n.

From the argumentation above it follows that all elements of P are checked whether that element is nonzero, and thus all elements of $PR(:,j) \cdot PR(:,j)^T$ are checked. Therefore, as $j \in J_n$, the matrix $PR(:,j) \cdot PR(:,j)^T$ should contain at least one nonzero element. As each element in PR is either zero or one, we can conclude that $PR(:,j)^T \cdot PR(:,j) \geq 1$. ∎

D. Complete algorithm

Algorithm 3 shows the steps performed to transform a general problem definition into a tree-structured system with

at each node a plant model and a requirement model. The following theorem states that the result of Algorithm 3 is a valid input for synthesis of a set S_s of supervisors according to [32] and Theorem 1. Algorithm 3 also works correctly for FA models, as for those models $Var(G_i) = Var(K_j) = \emptyset$ for all G_i and K_j by definition.

Algorithm 3 TransformToMLDES

Input: set of plant models $\{G_i \mid i \in I\}$, set of requirement models $\{K_j \mid j \in J\}$

Output: index set of tree-structure T of MLDES, set of plant models $G_T = \{G_n \mid n \in T\}$, set of requirement models $K_T = \{K_n \mid n \in T\}$

1: Transform $\{G_i\}$ to a most refined product system $\{G'_i\}$ with new index set I'.
2: Construct matrix PR such that $PR(i,j) = 1$ iff $Alp(G'_i) \cap Alp(K_j) \neq \emptyset \vee Var(G'_i) \cap Var(K_j) \neq \emptyset, \forall i \in I', j \in J$.
3: Calculate $P = PR \cdot PR^T$.
4: Cluster P, for example with algorithm presented in [42]. Assign the computed multilevel clustering to (I', M).
5: Transform (I', M) to the tree structure including $\{G_n\}$, $\{K_n\}$ by Algorithm 1 with initial parameter values Transform$C^m$to$T((I', M), 1, P, PR, \{G'_i\}, \{K_j\})$

Theorem 2 (Valid MLDES tree): Consider a general composed system $\{G_i \mid i \in I\}$ and prefix-closed requirements $\{K_j \mid j \in J\}$ with $G = \|_{i \in I}\ G_i$ and $K = \|_{j \in J}\ K_j$, respectively. The output $T, \{G_n \mid n \in T\}$, and $\{K_n \mid n \in T\}$ generated by Algorithm 3 is an MLDES for synthesis of the set $S_s = \{S_n \mid n \in T, S_n \parallel G_n \subseteq K_n\}$ according to Theorem 1.

Proof: It suffices to show that $G = \|_{n \in T}\ G_n,\ K = \|_{n \in T}\ K_n$ and $\forall n \in T : K_n \subseteq \Sigma^*_{G_n}$. Theorems 3 and 4 show that $G = \|_{n \in T}\ G_n$ and $K = \|_{n \in T}\ K_n$, respectively. It only remains to prove that $\forall n \in T : K_n \subseteq \Sigma^*_{G_n}$.

Consider a node n, it holds that $K_n \subseteq \Sigma^*_{G_n}$ if $\Sigma_{K_n} \subseteq \Sigma_{G_n}$. In Algorithm 2, K_n is constructed at Line 18 according to the index set J_n. One can observe at Lines 3 and 10 of Algorithm 2 that $\forall j \in J_n : \forall i \in I' \wedge PR(i,j) = 1 : i \in I_n$. By the construction of PR given in Line 2 of Algorithm 3 and the fact that $G_n = \|_{i \in I_n}\ G'_i$, we can conclude that $\Sigma_{K_n} \subseteq \Sigma_{G_n}$. ■

Theorem 2 applies for prefix-closed requirements. In case of non-prefix closed requirements, the presented method can still be used to construct an MLDES. After synthesizing supervisors for the MLDES, a nonconflicting check should be performed as mentioned in Section III-C.

Example *Fig. 6 shows the result after applying Algorithm 3. For each node $n \in T$, supervisor S_n is synthesized with monolithic supervisory control synthesis procedure. Table I shows the number of states and transitions of the supervisors. For a comparison, also the number of states and transitions are shown of the uncontrolled system and the supervisory controller obtained with monolithic synthesis procedure. We noticed in this example that all nodes at level 3 have no requirements and each plant model is already located in one of the nodes where a supervisor is synthesized, so no supervisors are synthesized at these nodes as any potential blocking state in the plant models is solved by one of the four supervisors.*

As can be seen, the constructed set of supervisors $\{S_n \mid n \in T\}$ has a smaller automaton representation than a single monolithic supervisor S_{mono}. Furthermore, it has been checked that the system controlled by the set of supervisors is nonblocking. Therefore, it holds that $\|_{n \in T}\ S_n = S_{mono}$.

TABLE I
RESULTS OF SUPERVISORY CONTROL SYNTHESIS ON THE SIMPLE LOCK MODEL

Synthesis architecture	Subsystem	Number of states	Number of transitions
Uncontrolled system	G	82.944	1.041.408
Monolithic supervisor	S_{mono}	688	4288
MLDES supervisors	Sum	227	954
	$S_{(1,1,1)}$	176	824
	$S_{(2,1,1)}$	22	59
	$S_{(2,1,2)}$	22	59
	$S_{(2,1,3)}$	7	12

E. Complexity analysis

The complexity of Algorithm 3 and performing subsequently multilevel synthesis depends on the reduction in size of the problem definition in each node. In the worst case, the most refined product system reduces the original set of plants to only a single plant model, resulting in monolithic synthesis. Nevertheless, in [32] it is shown that for some MLDES a considerable gain in computational time complexity of supervisor synthesis can be achieved.

In the next section, several models from the literature are analyzed to assess the applicability of the presented method. The purpose is (1) to determine whether the above described worst-case scenario is present in models of realistic systems and (2) to observe the complexity reduction for different scenarios.

V. EXPERIMENTAL RESULTS

The presented method to transform any problem definition into an MLDES using DSMs has been implemented in the discrete-event systems tool CIF [51] and Matlab [52]. Three different experiments have been executed. First, the effect of different values for clustering parameters on the synthesized supervisors has been investigated. Second, the effect of different clusterings on nonblocking control has been investigated. Finally, tests on benchmark models have been performed. The CIF representation of the models used and results generated can be found in [50].

A. Clustering parameters

The first experiment investigates the observed computational complexity by applying the proposed method with different clustering settings. We chose $\alpha \in \{1, 2, 3\}$, $\beta \in$

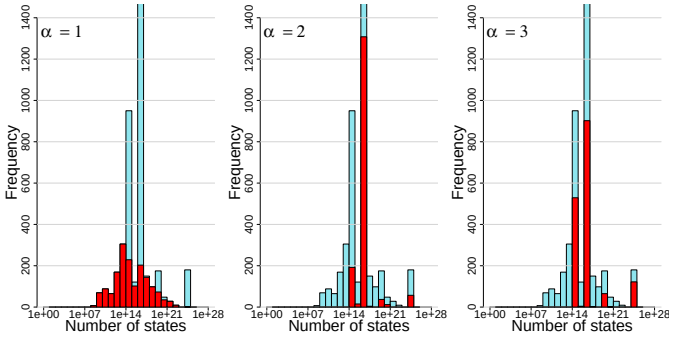

Fig. 7. Histograms in red of sum of controlled state space sizes split for $\alpha = 1$ (left), $\alpha = 2$ (middle), and $\alpha = 3$ (right). The blue histogram in the background is the complete histogram without splitting the data.

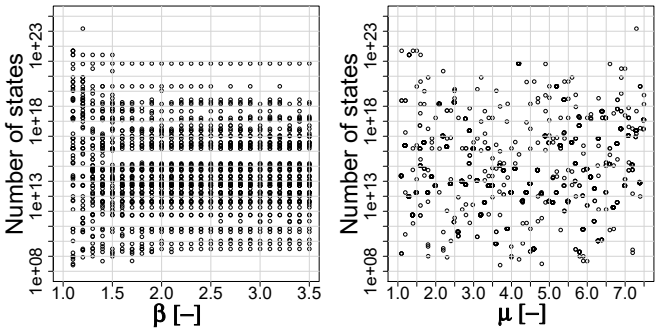

Fig. 8. Scatter plots of all data with $\alpha = 1$. On the left, the parameter β is placed on the horizontal axis, while on the right it is parameter μ. Both plots have the sum of controlled state space sizes on the vertical axis. In case several parameter settings result in the same quantitative index, the circles are drawn on top of each other.

$\{1.1, 1.2, \ldots, 3.5\}$, and $\mu \in \{1.1, 1.2, \ldots, 7.5\}$. The ranges are extended versions of the recommended ranges in [42]. As the goal is to synthesize supervisors, we chose the sum over the nodes in the MLDED tree of the controlled state space size, i.e., $\sum_{n \in T} mcss_n$, as the quantitative index to compare different clustering results (and no quantitative index based on the clustered DSM). We performed all experiments on the **lockIII** model.

lockIII A waterway lock is used in rivers and canals to raise and lower vessels between different water levels [38]. This model has various subsystems: gates, paddles, culverts, two-lamp traffic lights, and three-lamp traffic lights. An operator can interact with the system through a human-machine interface.

Fig. 7 shows three histograms with the results focussing on the effect of parameter α. On the horizontal axis, the quantitative index is shown. In this case, the closer to the origin, the better we consider the results. From these histograms, it can be conclude that for the **lockIII** model it is better to have $\alpha = 1$ instead of $\alpha = 2$ or $\alpha = 3$. We can also explain this result on a conceptual level. The parameter α is involved in the expansion step where the probability matrix of a Markov chain (representing the DSM) is raised to the power α, i.e., it indicates a random walker walking α steps. The DSM of the **lockIII** model is quite dense. Having an $\alpha > 1$ results in the ability that the random walker may reach (almost) all nodes. This reduces the ability to cluster, as everything seems to be directly connected to everything else.

Fig. 8 shows the effect of parameters β and μ. On the vertical axis, the quantitative index is shown. Again, the closer to the origin, the better we consider the results. First, we focus on the effect of parameter β (the left scatter plot). We may distinguish two regions in the scatter plot: approximately $\beta < 2$ and $\beta > 2$. For $\beta > 2$ we see repetitive results in the scatter plot, while for $\beta < 2$ we see both decrease of minimum and increase of maximum. So, the interesting region to search for the absolute minimum of the quantitative index is $\beta < 2$, but from this scatter plot it is still unclear if, for example, the smaller β is, the better are the synthesis results.

Second, we focus on the effect of parameter μ (the right scatter plot). The results are really scattered around and no distinct pattern can be detected.

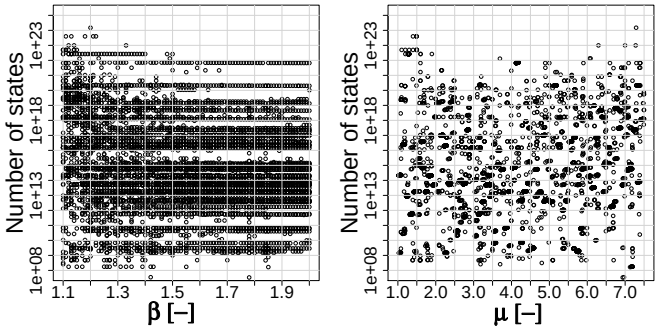

Fig. 9. Scatter plots of all data with $\alpha = 1$ and increased precision of β. On the left, the parameter β is placed on the horizontal axis, while on the right it is parameter μ. Both plots have the sum of controlled state space sizes on the vertical axis. In case several parameter settings result in the same quantitative index, the circles are drawn on top of each other.

Looking at these results from an engineering perspective, we can conclude the following. As an engineer, you may be satisfied with a good reduction and not primarily the best reduction. For **lockIII**, the monolithic controlled state space size is $6.0 \cdot 10^{24}$, so reducing this to $10^9 - 10^{10}$ is already a good reduction, while $\sim 10^8$ would be the best reduction. No matter which β is chosen, there exist multiple values of μ that would result in a good reduction. This manual search for a good reduction turns out to be easy.

Finally, we were also interested in the effect of parameter β in the region $\beta < 2$. Therefore, we repeated the above experiment for $\alpha = 1$, $\beta \in \{1.1, 1.11, \ldots, 2.0\}$, and $\mu \in \{1.1, 1.2, \ldots, 7.5\}$. The results are shown in Fig. 9. Two parameter settings having a quantitative index smaller than 10^8 can now be identified, which were not observed in Fig. 8. Nevertheless, no clear relationship between the parameter values of β and μ and the quantitative index can be observed. So, if the best reduction is desired, it may involve more effort.

B. Nonblocking control

The second experiment investigates the effect of different clustering outcomes on nonblocking control. Again, the **lockIII** model is used for these experiments. The model does

not contain any conflicting requirements, so any clustering and MLDES would result in nonconflicting supervisors. Therefore, a conflicting requirement is introduced into the model. For this new model, monolithic synthesis would result in nonblocking control, while modular supervisors [10] would result in conflicting supervisors.

The following approach is used to create a conflicting requirement. For requirement $R_1 = e$ **needs** C, which expresses that event e is only allowed if condition C is satisfied, a new potential conflicting requirement is $R_2 = e$ **needs** $\neg C$. The combination of these two requirements expresses that event e is never allowed to happen. Blocking event e is problematic if a non-marked state can be reached having an outgoing transition labeled with e and this transition must be included in any path to a marked state.

Above strategy is insufficient for this experiment: both requirements $R_1 = e$ **needs** C and $R_2 = e$ **needs** $\neg C$ have the same dependencies with plant models (the plant model of e and those used in C). Therefore, these two requirements are always placed in the same node and synthesis will resolve the blocking issue in that node. Therefore, the condition of the artificial requirement is extended as follows: $R_3 = e$ **needs** $\neg C \wedge (D \vee \neg D)$ with D another condition. It is easy to see that $\neg C$ and $\neg C \wedge (D \vee \neg D)$ are logically equivalent. Therefore, this extended requirement results in the same blocking issue as with the first artificial requirement. We now use condition D to relate this extended requirement R_3 to another plant model not yet mentioned in the original requirement R_1. This opens the possibility that both requirements end up in different nodes in the MLDES.

We applied the above strategy to create an artificial blocking requirement in **lockIII**. Conflicting supervisors are obtained with clustering settings $\alpha = 1$, $\beta = 2.0$, and $\mu = 4.6$, and nonconflicting supervisors with $\alpha = 1$, $\beta = 2.0$, and $\mu = 4.5$. In the first clustering, the two conflicting requirements are placed in different nodes, while in the latter they are placed in the same node. These results show that the proposed method may lead to a solution for nonblocking control, but it should be further investigated under which conditions the nonblocking result can be guaranteed.

C. Benchmark testing

The implementation of the presented algorithm has been tested on several models available in the literature, which are listed below. The models selected are either fully described in the reference or fully available in the discrete-event systems tool Supremica [53]. Most of the models have also been used previously in benchmark testing, see, e.g., [7], [35], [46].

transfer-line In this transfer line, products are processed by two machines [45]. The first machine places the products after processing them in the first buffer. The second machine takes products from the first buffer and places them in the second buffer. A test unit verifies whether the products in the second buffer are acceptable. If accepted, the products leave the system, otherwise the products move back to the first buffer.

circular-table In this manufacturing cell metal pieces are drilled and tested [54]. The machine consists of a four-stages circular table with four operational devices: an input conveyer, a drilling machine, a test device, and a manipulator.

intertwined This models a manufacturing system where two types of products are processed [17]. The system consists of two machines, four handling devices, and six buffers. For each type of product, a pre-specified production cycle is given.

agent-formation This multi-agent formation problem considers a circular route where three agents can only travel clockwise [24], [45]. There are two alternative desired formations possible: an equilateral triangle and an alignment curve. A team leader or a remote operator decides which formation is currently needed.

work-cycle This manufacturing system consists of three machines and two buffers [49]. Parts are supplied through an input buffer and are after processing stored in two output buffers. In this system, the first buffer has a capacity of 16 products and the second buffer a capacity of 8 products.

agv A set of five automatic guided vehicles (AGVs) serve several workstations in a manufacturing cell [20]. Each AGV travels on one of the fixed circular routes serving two input stations, three work stations, and a single output station.

central-lock This models the central locking system of a BMW car. The modeled system consists of three doors controlled by a central locking system. The model is derived from the KorSys project and it is available in the tool Supremica [53].

cluster-tool An integrated manufacturing system that is used for processing wafers [22]. Four robots move the wafers in the system that consists of nine work chambers, three buffers, an input load lock, and an output load lock. Wafers need to be processed according to a pre-specified routing.

production-cell In this production cell, metal blanks need to be forged by a press [55]. The feed belt forwards blanks from the stock to the elevating rotary table. The first arm of the robot picks up the blank and places it in the press. After processing, the second arm from the same robot picks the blank and drops it on the deposit belt. At the end of the deposit belt, the test unit checks whether the forging was successful. If it passes the test, the blank leaves the system, otherwise it is moved back to the feed belt by the crane.

adas A car is modeled with two Advanced Driver Assistance Systems (ADASs): Cruise Control (CC) and Adaptive Cruise Control (ACC) [56]. CC is used to maintain a desired velocity using feedback control. ACC is used to maintain a constant inter-vehicle time gap with respect to the predecessor. The user operates the ADASs with a human-machine interface and can therefore choose between manual control, CC, or ACC.

testbed-rail A railroad system is used to resemble a set of three interacting work units in a manufacturing cell [57]. The trains simulate AGVs that handle material to and from each work unit. Each unit has a small crane for loading and unloading the material. Several switches in the track allow for different train paths.

wafer-scanner This model concerns the routing of wafers through a wafer scanner [58]. The system consists of two areas: the wafer stage where wafers are measured and exposed, and the wafer handler where several pre-exposure steps are

performed. The wafers enter and exit through the wafer handler area. Furthermore, there are two dummy wafers available in the system.

container-terminal A LEGO model of a container terminal system is used to demonstrate model-based engineering [59]. The system consists of three lanes each with a moving crane and a truck transporting containers between the three lanes. Containers are loaded into the system in one lane and finally unloaded via one of the other two lanes. The choice between one of the two unload lanes depends on the color of the container.

festo This didactic production line system consists of 28 actuators and 59 sensors [60]. Products undergo various processing steps in six different workstations: distributing station, handling station, testing station, buffering station, processing station, and sorting station.

For each model we collect the following metrics: the modeling formalism, the number of plant models $|I|$, the number of requirement models $|J|$, the number of plant models in the most refined product system $|I'|$, the uncontrolled state space size uss, the controlled state space size of the monolithic supervisor $mcss$, the sum of the controlled state space sizes of the multilevel supervisors $mlcss = \sum_{n \in T} mcss_n$, the number of supervisors ns, the calculation time t in seconds, and whether the calculated set of supervisors is nonblocking. For each individual model, a good clustering has been searched manually. The used clustering parameter values can be found in [50].

Table II shows the experimental results for the different models. The models are ordered based on the uncontrolled state space size. For some models, during the calculation of some metrics an out-of-memory (OoM) error has been encountered and indicated in Table II. A nonblocking check has been performed for those models that have non-prefix closed requirements. The table shows a wide range of diversity of the different models. All calculations have been performed on an HP ZBook laptop with Intel i7 2.4 GHz CPU and 8GB RAM. At most 2 GB of the available RAM could be allocated for the calculations.

The models can be clustered into two groups based on the experimental results: the smaller models and the larger models. The smaller models are the first 6 models (transfer-line up to and including agv); the larger models are the last 10 models (central-lock up to and including lockIII).

As expected, the benefit of multilevel synthesis is minimal or sometimes even absent for the smaller models. The number of independent plant models (indicated by $|I'|$) is often too small to allow for the creation of an effective MLDES. Only for the circular-table model an MLDES can be created that reduces the state space size of the multilevel supervisors. For the agv model, an MLDES is not beneficial. The state space size of the monolithic supervisor is already larger than the state space size of the uncontrolled system. Dividing requirements into different nodes even further increases the controlled state space size.

For the larger models, the benefit of MLDESs becomes more apparent. For example, for the two largest models (festo and lockIII), the state space size of the multilevel supervisors is significantly smaller than the state space size of the monolithic supervisor.

The worst-case scenario as described in Section IV-E becomes reality for some of the presented models. The cluster-tool as well as the wafer-scanner have a single plant in the most refined product system representation ($|I'| = 1$). For the wafer-scanner it was already reported in [58] that monolithic synthesis was not feasible. The testbed-rail resulted in only 4 plant models in the most refined product system representation, of which one contains almost all the original plant models. For this model, neither monolithic nor multilevel supervisors could be synthesized.

Creating an MLDES tackles the out-of-memory problem for the container model. While no monolithic supervisor could be synthesized, six multilevel supervisors are synthesized. MLDESs can be beneficial in case the synthesis of a monolithic supervisor turns out to be infeasible.

Finally, the formulation of the requirements has influence on the result of transforming a DES to an MLDES. For example, one could have the state-transition exclusion requirement σ **needs** $A.l_1 \wedge B.l_2 \wedge C.l_3$ stating that event σ is only allowed when automaton A is in location l_1 and automaton B is in location l_2 and automaton C is in location l_3. As the synthesized supervisor should satisfy *all* requirements, we can rewrite this single requirement as a set of three requirements: σ **needs** $A.l_1$, σ **needs** $B.l_2$, and σ **needs** $C.l_3$. Each of these three new requirements relates to a smaller set of plant models than the original one, resulting in fewer relationships in the DSM P, and finally locating these requirements in a lower node in the MLDES. This creates the potential of obtaining a smaller state space size of the multilevel supervisors.

To demonstrate this observation, the original adas model is reformulated into adas* where requirements are split into multiple smaller ones when possible. For the original adas model we obtained a state space size of $1.1 \cdot 10^8$ states. By splitting the requirements we can reduce the state space size to $5.2 \cdot 10^5$ states. Also observe that the number of supervisors is doubled from 8 to 16.

VI. Conclusion

In this paper, a framework is presented that can be used to transform a set of plant models and a set of requirement models into an MLDES. The plant models and requirement models are represented by EFAs and state-based expressions. Relationships between plant models and requirement models are used to analyze the structure in the problem description with DSM-based techniques in order to find an MLDES. For each node in the MLDES a supervisor can be synthesized.

Experimental results obtained for a set of models from the literature show that the presented approach helps in overcoming the state space explosion problem. For some models for which no monolithic supervisor could be synthesized, multilevel supervisors could be synthesized after transforming these models into an MLDES.

The benefit of MLDESs depends on the modeling decisions made by the modeler in creating the model. First, when the plant models are 'loosely coupled', i.e., only a few plant

TABLE II
EXPERIMENTAL RESULTS FOR DIFFERENT MODELS

Model	Form.	$\|I\|$	$\|J\|$	$\|I'\|$	uss	$mcss$	$mlcss$	ns	t	Nonblocking
transfer-line	FA	3	2	3	8	28	30	2	4.5	no
circular-table	FA	5	8	5	32	151	91	2	4.8	yes
intertwined	FA	6	6	2	256	9,216	9,248	3	4.5	yes
agent-formation	EFA	4	13	1	1,000	304	304	1	0.27	yes
work-cycle	EFA	5	2	1	1,224	1,188	1,188	1	0.02	yes
agv	FA	5	8	5	1,664	4,406	9,837	3	4.9	no
central-lock	FA	74	35	74	$2.6 \cdot 10^5$	OoM	$3.9 \cdot 10^5$	15	12.0	yes
cluster-tool	FA	18	16	1	$2.6 \cdot 10^8$	$2.7 \cdot 10^6$	$2.7 \cdot 10^6$	1	7.8	yes
production-cell	FA	11	19	10	$3.8 \cdot 10^8$	$1.1 \cdot 10^8$	22,827	7	9.8	no
adas	EFA	28	33	27	$3.4 \cdot 10^9$	$2.0 \cdot 10^{10}$	$1.1 \cdot 10^8$	8	5.5	yes
adas*	EFA	28	72	27	$3.4 \cdot 10^9$	$2.0 \cdot 10^{10}$	$5.2 \cdot 10^5$	16	5.7	yes
testbed-rail	FA	6	29	4	$5.6 \cdot 10^9$	OoM	OoM	OoM	OoM	OoM
wafer-scanner	EFA	48	37	1	OoM	OoM	OoM	OoM	OoM	OoM
container-terminal	EFA	45	35	15	$3.8 \cdot 10^{22}$	OoM	$3.4 \cdot 10^{18}$	6	275	yes
festo	EFA	113	211	88	$1.5 \cdot 10^{26}$	$2.2 \cdot 10^{25}$	50,638	24	9.6	yes
lockIII	EFA	71	198	51	$6.0 \cdot 10^{32}$	$6.0 \cdot 10^{24}$	$3.1 \cdot 10^9$	30	14.9	yes

models need to be combined to get the most refined product system, the benefit of MLDESs is very clear. For the festo model, 24 supervisors are created to reduce the controlled state space size of the monolithic supervisor of $2.2 \cdot 10^{25}$ states to only $50,638$ states for the multilevel supervisors. Second, splitting a composite requirement into multiple separate, but together equivalent, requirements can also benefit the result, as shown with the adas model.

Future research directions include specifying modeling guidelines to tackle the above mentioned problems. For instance, it may be possible to define certain rewriting algorithms like a 'requirement decomposition' algorithm. In case the model cannot be rewritten (easily), one may need to use other clustering architectures like, for example, overlapping clusters. Another direction is to consider other supervisory control architectures, like hierarchical and coordinated supervisory control, and investigate the changes needed to the proposed method such that proper input is generated for those architectures.

Furthermore, the question remains whether a different structure analysis could guarantee nonblocking control of MLDES. Finally, DSM-based structure analysis may also be useful for guiding abstraction-based methods like coordination control and compositional synthesis.

ACKNOWLEDGMENT

The authors thank prof.em. J.H. van Schuppen for several fruitful discussions about the control of MLDESs and his comments on a manuscript of this paper.

REFERENCES

[1] P. J. G. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[2] ——, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[3] M. Skoldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *46th IEEE Conf. on Decision and Control*, Dec. 2007, pp. 3387–3392.

[4] S. Miremadi, K. Åkesson, B. Lennartson, and M. Fabian, "Supervisor computation and representation: a case study," in *10th Int. Workshop on Discrete Event Systems*, Berlin, Germany, 2010, pp. 275–280.

[5] C. Ma and W. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, ser. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2005, no. 317.

[6] J. Markovski, K. G. M. Jacobs, D. A. van Beek, L. J. Somers, and J. E. Rooda, "Coordination of resources using generalized state-based requirements." 2010, pp. 300–305.

[7] Z. Fei, S. Miremadi, K. Åkesson, and B. Lennartson, "Efficient symbolic supervisor synthesis for extended finite automata," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2368–2375, Nov. 2014.

[8] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of Control, Signals and Systems*, vol. 1, no. 1, pp. 13–30, Feb. 1988.

[9] K. C. Wong and W. M. Wonham, "Modular Control and Coordination of Discrete-Event Systems," *Discrete Event Dynamic Systems*, vol. 8, no. 3, pp. 247–297, Oct. 1998.

[10] M. H. d. Queiroz and J. E. R. Cury, "Modular Supervisory Control of Large Scale Discrete Event Systems," in *Discrete Event Systems*, ser. The Springer International Series in Engineering and Computer Science, R. Boel and G. Stremersch, Eds. Springer US, 2000, no. 569, pp. 103–110.

[11] K. Åkesson and M. Fabian, "Exploiting Modularity for Synthesis and Verification of Supervisors," in *15th Triennial World Congress*, 2002.

[12] M. Teixeira, J. E. R. Cury, and M. H. d. Queiroz, "Local modular Supervisory Control of DES with distinguishers," in *16th IEEE Conf. on Emerging Technologies and Factory Automation*, Sep. 2011, pp. 1–8.

[13] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Trans. Automat. Contr.*, vol. 35, no. 10, pp. 1125–1134, Oct. 1990.

[14] J. G. Thistle, "Logical aspects of control of discrete-event systems: A survey of tools and techniques," in *11th Int. Conf. on Analysis and Optimization of Systems Discrete Event Systems*, ser. Lecture Notes in Control and Information Sciences, G. Cohen and J.-P. Quadrat, Eds. Springer Berlin Heidelberg, 1994, no. 199, pp. 1–15.

[15] K. C. Wong and W. M. Wonham, "Hierarchical control of discrete-event

systems," *Discrete Event Dynamic Systems*, vol. 6, no. 3, pp. 241–273, Jul. 1996.

[16] R. J. Leduc, P. Dai, and R. Song, "Synthesis Method for Hierarchical Interface-Based Supervisory Control," *IEEE Trans. Automat. Contr.*, vol. 54, no. 7, pp. 1548–1560, Jul. 2009.

[17] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE Trans. Automat. Contr.*, vol. 35, no. 12, pp. 1330–1337, Dec. 1990.

[18] K. Rudie and W. M. Wonham, "Think globally, act locally: decentralized supervisory control," *IEEE Trans. Automat. Contr.*, vol. 37, no. 11, pp. 1692–1708, Nov. 1992.

[19] T. S. Yoo and S. Lafortune, "A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems," *Discrete Event Dynamic Systems*, vol. 12, no. 3, pp. 335–377, Jul. 2002.

[20] L. Feng and W. M. Wonham, "Supervisory Control Architecture for Discrete-Event Systems," *IEEE Trans. Automat. Contr.*, vol. 53, no. 6, pp. 1449–1461, Jul. 2008.

[21] K. Cai and W. M. Wonham, "Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems," *IEEE Trans. Automat. Contr.*, vol. 55, no. 3, pp. 605–618, Mar. 2010.

[22] R. Su, J. H. van Schuppen, and J. E. Rooda, "Aggregative Synthesis of Distributed Supervisors Based on Automaton Abstraction," *IEEE Trans. Automat. Contr.*, vol. 55, no. 7, pp. 1627–1640, Jul. 2010.

[23] C. Seatzo, M. Silva, and J. H. van Schuppen, *Control of Discrete Event Systems - Automata and Petri Net Perspectives*, ser. Lecture Notes in Computer Science. London: Springer, 2013, no. 433.

[24] K. Cai and W. M. Wonham, "New results on supervisor localization, with case studies," *Discrete Event Dynamic Systems*, vol. 25, no. 1-2, pp. 203–226, May 2014.

[25] R. Zhang, K. Cai, Y. Gan, and W. M. Wonham, "Distributed supervisory control of discrete-event systems with communication delay," *Discrete Event Dynamic Systems*, vol. 26, no. 2, pp. 263–293, Jan. 2015.

[26] J. Komenda and J. H. van Schuppen, "Coordination control of discrete-event systems," in *9th Int. Workshop on Discrete Event Systems*, May 2008, pp. 9–15.

[27] J. Komenda, T. Masopust, and J. H. van Schuppen, "Supervisory control synthesis of discrete-event systems using a coordination scheme," *Automatica*, vol. 48, no. 2, pp. 247–254, Feb. 2012.

[28] ——, "Coordination control of discrete-event systems revisited," *Discrete Event Dynamic Systems*, vol. 25, no. 1-2, pp. 65–94, Feb. 2014.

[29] R. Hill and D. Tilbury, "Modular supervisory control of discrete-event sytems with abstraction and incremental hierarchical construction," in *8th International Workshop on Discrete Event Systems*, Jul. 2006, pp. 399–406.

[30] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional Synthesis of Maximally Permissive Supervisors Using Supervision Equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, Dec. 2007.

[31] R. Malik and H. Flordal, "Yet another approach to compositional synthesis of discrete event systems," in *9th International Workshop on Discrete Event Systems*, May 2008, pp. 16–21.

[32] J. Komenda, T. Masopust, and J. H. van Schuppen, "Control of an engineering-structured multilevel discrete-event system," in *13th Int. Workshop on Discrete Event Systems*, May 2016, pp. 103–108.

[33] S. D. Eppinger and T. R. Browning, *Design structure matrix methods and applications*. MIT press, 2012.

[34] T. R. Browning, "Design structure matrix extensions and innovations: a survey and new opportunities," *IEEE Trans. Eng. Manage.*, vol. 63, no. 1, pp. 27–52, 2016.

[35] H. Flordal and R. Malik, "Compositional Verification in Supervisory Control," *SIAM Journal on Control and Optimization*, vol. 48, no. 3, pp. 1914–1938, Jan. 2009.

[36] J. Komenda, T. Masopust, and J. H. van Schuppen, "Multilevel Coordination Control of Modular DES," in *52th IEEE Conf. on Decision and Control*, Florence, Italy, Dec. 2013, pp. 6323–6328.

[37] L. Feng and W. M. Wonham, "Computationally Efficient Supervisor Design: Control Flow Decomposition," in *8th Int. Workshop on Discrete Event Systems*, Jul. 2006, pp. 9–14.

[38] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory control synthesis for a waterway lock," in *1st IEEE Conf. on Control Technology and Applications*, Aug. 2017, pp. 1562–1568.

[39] M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, and J. E. Rooda, "Structuring multilevel discrete-event systems with dependency structure matrices," in *56th IEEE Conf. on Decision and Control*, Dec. 2017, pp. 558–564.

[40] M. S. Maurer, "Structural awareness in complex product design," Ph.D. dissertation, Universität München, 2007.

[41] A. Yassine, D. Whitney, S. Daleiden, and J. Lavine, "Connectivity maps: Modeling and analysing relationships in product development processes," *Journal of Engineering Design*, vol. 14, no. 3, pp. 377–394, Sep. 2003.

[42] T. Wilschut, L. F. P. Etman, J. E. Rooda, and I. J. B. F. Adan, "Multilevel flow-based Markov clustering for design structure matrices," *Journal of Mechanical Design*, vol. 139, no. 12, p. 121402, 2017.

[43] S. van Dongen, "Graph clustering via a discrete uncoupling process," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 121–141, Jan. 2008.

[44] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Boston: Springer, 2008.

[45] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, ser. Communications and Control Engineering. Cham: Springer International Publishing, 2019.

[46] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *Discrete Event Dynamic Systems*, vol. 26, no. 1, pp. 33–84, Mar. 2016.

[47] R. Kumar, V. Garg, and S. I. Marcus, "Predicates and predicate transformers for supervisory control of discrete event dynamical systems," *IEEE Trans. Automat. Contr.*, vol. 38, no. 2, pp. 232–247, Feb. 1993.

[48] G. Birkhoff, *Lattice theory*. American Mathematical Soc., 1940, vol. 25.

[49] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. on Automat. Sci. and Eng.*, vol. 8, no. 3, pp. 560–569, Jul. 2011.

[50] M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, and J. E. Rooda, "CIF3 models used in the T-AC 2018 paper." [Online]. Available: https://github.com/magoorden/T-AC2018

[51] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers, "CIF 3: Model-based engineering of supervisory controllers," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Apr. 2014, pp. 575–580.

[52] Mathworks, "Matlab." [Online]. Available: https://www.mathworks.com/products/matlab.html

[53] R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica-An efficient tool for large-scale discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5794–5799, Jul. 2017.

[54] M. H. de Queiroz and J. E. R. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell," in *6th Int. Workshop on Discrete Event Systems*, 2002, pp. 377–382.

[55] L. Feng, K. Cai, and W. M. Wonham, "A structural approach to the non-blocking supervisory control of discrete-event systems," *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 11-12, p. 1152, Apr. 2009.

[56] T. Korssen, V. Dolk, J. M. van de Mortel-Fronczak, M. A. Reniers, and M. Heemels, "Systematic model-based design and implementation of supervisors for advanced driver assistance systems," *IEEE Trans. Intell. Transport. Syst.*, vol. 19, no. 2, pp. 533–544, 2017.

[57] R. J. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: an implementation perspective," Master's thesis, Dept. of Elect. Eng., Univ. of Toronto, Toronto, 1996.

[58] L. J. van der Sanden, M. A. Reniers, M. C. W. Geilen, A. A. Basten, J. Jacobs, J. P. M. Voeten, and R. R. H. Schiffelers, "Modular model-based supervisory controller design for wafer logistics in lithography machines," *Proc. 18th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*, 2015.

[59] M. A. Reniers and J. M. van de Mortel-Fronczak, "An engineering perspective on model-based design of supervisors," in *14th Int. Workshop on Discrete Event Systems*, May 2018, pp. 268–275.

[60] F. F. H. Reijnen, M. A. Goorden, J. M. v. d. Mortel-Fronczak, M. A. Reniers, and J. E. Rooda, "Application of Dependency Structure Matrices and Multilevel Synthesis to a Production Line," in *2018 IEEE Conf. on Control Technology and Applications*, Aug. 2018, pp. 458–464.

Martijn Goorden received the M.Sc. degree (*cum laude*) in systems and control from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2015. He is currently working towards the Ph.D. degree in mechanical engineering.

His current research interests are in the area of model-based systems engineering and supervisory control synthesis.

Joanna van de Mortel-Fronczak received the M.Sc. degree in computer science from AGH University of Science and Technology, Cracow, Poland, in 1982 and the Ph.D. degree in computer science from Eindhoven University of Technology, Eindhoven, The Netherlands, in 1993.

Since 1997, she has been with the Department of Mechanical Engineering, Eindhoven University of Technology. Her research interests include model-based engineering and the synthesis of supervisory control systems.

Michel Reniers (S'17) is currently an Associate Professor in model-based engineering of supervisory control at the Department of Mechanical Engineering, Eindhoven University of Technology. He has authored over 100 journal and conference papers, and is the supervisor of ten Ph.D. students. His research portfolio ranges from model-based systems engineering and model-based validation and testing to novel approaches for supervisory control synthesis. Applications of this work are mostly in the areas of high-tech systems and cyber-physical systems.

Wan Fokkink received his Ph.D. degree in Computer Science from the University of Amsterdam, Amsterdam, The Netherlands.

Since 2004 he is full professor of Theoretical Computer Science at the Vrije Universiteit Amsterdam. Since 2012 he is moreover professor of Model-Based System Engineering at Eindhoven University of Technology.

His research focus is on the design and analysis of distributed computer systems.

Jacobus Rooda received the M.Sc. degree from Wageningen University of Agriculture Engineering, Wageningen, The Netherlands, and the Ph.D. degree from Twente University, Enschede, The Netherlands.

Since 1985, he has been a Professor of (Manufacturing) Systems Engineering at the Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands.

Since 2010, he is a Professor Emeritus. He is still active in the research fields of engineering design for industrial systems, and of supervisory control thereof.