Structuring Multilevel Discrete-Event Systems with Dependency Structure Matrices

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

Abstract— **In this paper, we present a systematic approach to transform a set of plant models and requirement models into a tree-structured multilevel discrete-event system to which multilevel supervisory controller 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. Since one of the major drawbacks of synthesizing supervisory controllers is state space explosion, multiple attempts exist to overcome this computational difficulty, such as modular, hierarchical, decentralized, and, recently developed, multilevel supervisory control synthesis. Unfortunately, the modeler needs to provide additional information as input for most of these non-monolithic synthesis procedures. For those supervisory control synthesis procedures that require additional information, no systematic approach exists in literature to transform any set of plant models and requirement models to the appropriate input needed for a particular synthesis procedure. The presented approach is applied to a model of a lock and a model of an MRI scanner patient support table.**

I. Introduction

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

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, as already observed in [2]. Several attempts exploiting different architectures are proposed to overcome these computational difficulties: modular [3], hierarchical [4], decentralized [5], distributed [6], coordinated [7], and, more recently, multilevel supervisory control synthesis [8].

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

[1]Martijn Goorden (m.a.goorden@tue.nl), Joanna v.d. Mortel-Fronczak, Michel Reniers, and Jacobus Rooda are with Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands.

A problem with several of these supervisory control architectures is that additional information needs to be provided as input for synthesis. For example, hierarchical supervisory control needs an information mapping between the different levels, decentralized control requires projections to the subsystem alphabets, and multilevel control needs a tree-structured system. For those supervisory control synthesis procedures that require additional information, no systematic approach exists 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, the problem definition includes the specifications of both the plant models and the requirement models.

In this paper, we show how to exploit the problem definition structure by using Dependency Structure Matrices (DSMs) to transform the problem definition into an appropriate tree structure needed for multilevel supervisory control of [8]. A DSM provides a concise representation for the analysis of the structure of systems in many areas of research [9], [10]. With appropriate analysis techniques, one is able to highlight important aspects in system architectures, such as modules and cycles.

In contrast to suggestions found in literature [11], we analyze the relationship between the plant models and requirement models instead of relationships just between plant models (e.g., shared events). Our motivation is that we want to perform supervisory control synthesis which requires both plant models and requirement models.

The contribution of this paper is a proposal for a systematic approach to transform a problem definition into a tree-structured multilevel discrete-event system with the properties needed for multilevel supervisory control synthesis of [8]. This transformation uses DSM techniques to analyze the relationships present in the provided problem definition.

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. In Section V, the presented method is applied to an industrial case. The conclusions are presented in Section VI.

II. Dependency Structure Matrices

In this section, the concepts and notations of *Dependency Structure Matrices* (also called Design Structure Matrices) used in this paper are summarized. A more in depth introduction to DSM analysis is given in [9]. Examples and applications of DSMs can be found in the recent review paper [10].

A DSM is a square matrix with the same entities, or elements, along its axis (e.g., components in a system) and cells representing relationships between the entities (e.g., a spatial relationship). This relationship can be different per DSM.

There exist different types of DSMs. Undirected relationships result in a static DSM, while directed relationships result in a dynamic DSM. A type of DSM in which the relationships between different domains are described is called a *Domain Mapping Matrix* (DMM), which is a rectangular matrix. The generation of DSMs from DMMs and DSMs of other domains are described in [12], [13].

The different types of DSMs allow for different types of analyses of the considered system. In this paper, a static DSM will be analyzed. The focus of the literature on static DSMs is to find a modular architecture by clustering the entities of the DSM, as shown, for example, in [14].

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 can be found in [15], [16].

A. Preliminaries

We assume that the reader is familiar with the concepts of languages over an alphabet, synchronous product, automata representation, and monolithic supervisory control synthesis. Notations used below are taken from [16].

A less frequently used notion is that of a composed system representation. A *composed system* is a system model where the global plant G is represented by a set of subplants $G_i, i \in I = \{1, 2, \ldots, n\}$ such that $G = \|_{i \in I}\ G_i$, with $\|$ being the synchronous product. A composed system is called a *product system* if the alphabets of any two subplants are disjoint [2]. The *most refined product system* of a composed system is the product system that requires the minimal computational effort on product operations to obtain a product system. In [17], a procedure is provided to transform a composed system into the most refined product system.

B. Multilevel discrete-event systems

A *multilevel discrete-event system* (MLDES) is a system with a tree-based structure, as recently first proposed in [8]. More formally, let T represent an index set for a tree structure, where each element $(t, u, v) \in T$ represents a node at level t, in group u, and member v of that group. For notational simplicity, we write $n = (t, u, v)$ to point to a node in the tree structure. 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 nonblockingness, safety, controllability, and maximal permissiveness. In this paper we refer to S as the automaton representation of the synthesized supervisor.

Fig. 1: The lock at Terneuzen, The Netherlands. Image from https://beeldbank.rws.nl, Rijkswaterstaat.

As shown in [8], 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.

Theorem 1 (Existence of MLDES supervisors [8]): Consider an MLDES with subplant set G_s and a set of prefix-closed 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 $S \parallel G = \|_{n \in T}\ S_n \parallel G_n$.

It can then also be shown that the solution is maximally permissive and nonblocking [8], [15]. The more general situation is still ongoing research.

IV. Proposed method

In this section, the proposed method of transforming a general problem definition into an MLDES is described. The input for the method can be any general problem definition. The transformation consists of three stages: recording the dependencies in the problem definition, finding a valid clustering of the composed system, and constructing the MLDES. These three stages are explained in detail below. Finally, the complete algorithm to transform a general problem definition into an MLDES is provided.

As a general problem definition we mean 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. 1 shows the lock located at Terneuzen, The Netherlands.*

The following subplants are present in this simplified system:

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

Fig. 2: The DMM PR of the simple lock. Only the nonzero elements are shown for readability.

- *Entering light side 1*
- *Leaving light side 1*
- *Door actuator side 1*
- *Door sensor side 1*
- *Sewer actuator side 1*
- *Sewer sensor side 1*
- *Equal water sensor side 1*
- *Entering light side 2*
- *Leaving light side 2*
- *Door actuator side 2*
- *Door sensor side 2*
- *Sewer actuator side 2*
- *Sewer sensor side 2*
- *Equal water sensor side 2*

On this system, 26 requirements are imposed to guarantee the safe operation of a lock. For example, if there is no equal water over a door, then the door may not be opened.

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 can create a DSM with plants as entities with simple matrix multiplications, see [12].

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

Let the DMM be denoted by PR. Construct PR such that $PR(i, j) = 1$ if the alphabets of component i and requirement j are not disjoint, else $PR(i, j) = 0$.

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

1) *Entering light side 1*
2) *Leaving light side 1*
3) *Door side 1*
4) *Sewer side 1*
5) *Equal water sensor side 1*
6) *Entering light side 2*
7) *Leaving light side 2*
8) *Door side 2*
9) *Sewer side 2*
10) *Equal water sensor side 2*

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. 2. Consider the requirement mentioned before: if there is no equal water over a door, then the door may not be opened.

P	1	2	3	4	5	6	7	8	9	10
1	5	2	3							
2	2	5	3							
3	3	3	11		1			2	2	
4				4				2	2	
5			1		1					
6						5	2	3		
7						2	5	3		
8			2	2		3	3	11		1
9			2	2					4	
10								1		1

P_C	1	2	3	5	6	7	8	10	4	9
1	5	2	3							
2	2	5	3							
3	3	3	11	1			2			2
5			1	1						
6					5	2	3			
7					2	5	3			
8			2		3	3	11	1	2	
10							1	1		
4							2		4	2
9			2						2	4

Fig. 3: DSM P for the simple lock example: left the unclusterd P and right the clustered P_C.

For side 1, this is the third requirement. This requirement has a relationship with the door component (number 3) and the equal water sensor component (number 5). Therefore, $PR(3, 3) = PR(5, 3) = 1$ *and all other elements in column 3 of* PR *are zero.*

B. Finding a valid clustering

Before we can find clusters, we need to formally define a multilevel clustering.

Definition 1 (Multilevel clustering): The set of all multilevel clusterings C_A^m on a non-empty element set A is inductively defined.

- When $|A| = 1$, $(A, A) \in C_A^m$
- Assume $(A_1, V_1), \ldots, (A_s, V_s)$ with $2 \leq s \leq |A|$ s.t. $\{A_1, \ldots, A_s\}$ is a partition of A and $\forall i, 1 \leq i \leq s$: $(A_i, V_i) \in C_{A_i}^m$, then $(A, \{(A_i, V_i) \mid 1 \leq i \leq s\}) \in C_A^m$.

A multilevel clustering can be seen as recursively partitioning the set A, i.e., the set A is partitioned into $\{A_1, \ldots, A_s\}$ where each partitioning is again partitioned and so on until a partitioning with a single element is reached. In the tuple $(A, V) \in C_A^m$, A provides immediately all elements in this multilevel clustering and the set V 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 $\{1, 2, 3\}$.

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

By creating the DSM P from the DMM PR as defined before, we have the following interpretation. When $P(a, b) = k$ we know that there exists k requirements that use events from both G'_a and G'_b to describe the desired behavior.

Example *Fig. 3 shows the DSM* P *of the lock example. For example, cell* $P(3, 1) = 3$ *indicates that there are 3 requirements which use both plant component 1 (entering light side 1) and plant component 3 (door 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 [14] is used to find a multilevel clustering. This paper does not discuss which clustering algorithm is the best for our purpose. Any valid multilevel clustering according to Definition 1 can be

used. The chosen algorithm provides a multilevel clustering which fits our definition, which can be easily observed from the clustering algorithm description in [14]. Furthermore, it does not require any information on the structure as input, like the number of expected clusters or the number of hierarchical layers.

Example *Fig. 3 shows the clustered DSM P_C for the simple lock model. There are three clusters indicated: a cluster for lock side 1, a cluster for lock 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', V) on the index set I' of the product system $\{G'_i \mid i \in I'\}$. We will use the information from the 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 requirements $\{K_n \mid n \in T\}$.

In this section, we present two algorithms to construct both the tree index set T and the problem definition at each node in the tree, respectively. 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: $T, (A, V), a, b, c, P, PR, \{G'_i\}, \{K_j\}, \{G_n\}, \{K_n\}$
Output: $T, b, \{G_n\}, \{K_n\}$
Ensure: Add a new node to T and identify the children

1: add $(a, b[a], c)$ to T
2: $G_{(a,b[a],c)}, K_{(a,b[a],c)}, P, PR =$ Calculate$G_n$and$K_n((A, V), P, PR, \{G'_i\}, \{K_j\})$
3: **if** length$(V) > 1$ **then**
4: $a = a + 1$
5: **if** length$(b) < a$ **then**
6: $b = b + [1]$
7: **else**
8: $b[a] = b[a] + 1$
9: **end if**
10: $c = 1$
11: **for all** $(A_p, V_p) \in V$ **do**
12: $T, b, \{G_n\}, \{K_n\} =$ Transform$C^m$to$T(T, (A_p, V_p), a, b, c, P, PR, \{G'_i\}, \{K_j\}, \{G_n\}, \{K_n\})$
13: $c = c + 1$
14: **end for**
15: **end if**

Algorithm 1 shows the algorithm to create the index set T of the tree structure embedded in the given multilevel clustering (I', V). We apply this algorithm recursively to do a depth-first search through the multilevel clustering. Initially, Algorithm 1 is called with Transform$C^m$to$T$ $(\emptyset, (I', V), 1, [1], 1, P, PR, \{G'_i | i \in I'\}, \{K_j | j \in J\}, \emptyset, \emptyset)$.

When Algorithm 1 is performed, we know that there exists a valid level in the tree structure. This level is added to the tree structure on Line 2. Furthermore, on Line 2 the plant model and requirement model on level $(a, b[a], c) = n$ is calculated, see Algorithm 2. The algorithm continues when we can go further down in the tree structure. Line 4-10 are used for bookkeeping of the new values of a, b and c. At Line 12 we go further down the tree structure for each subcluster.

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 4 child nodes, the second subcluster also has 4 child nodes, and the third subcluster has 2 child nodes. The resulting tree T has three levels and is shown in Fig. 4.*

Algorithm 2 Calculate$G_n$andK_n

Input: $(A, V), P, PR, \{G'_i\}, \{K_j\}$ **Output:** G_n, K_n, P, PR
Ensure: Determines the problem definition for the top node of the clustering (I, V)

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

Algorithm 2 shows the procedure to calculate the problem definition for a certain node n in the tree structure. $P(:, j)$ indicates the column vector j of matrix P. We need to make a distinction between a leaf node or a non-leaf node.

When we have reached a leaf node, Lines 1-3, we set G_n equal 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 3.

When we did not reach a leaf node, the multilevel clustering (A, V) consists of multiple subclusters. At the current node in the tree, we need to identify the requirements which combine the subclusters into this particular multilevel clustering (A, V). To this end, we search in the DSM P nonzero elements outside each of the subclusters, but inside cluster (A, V). In Lines 5-7 we consider all possible combinations of elements from two different subclusters. When we find a nonzero element in P, we know that there

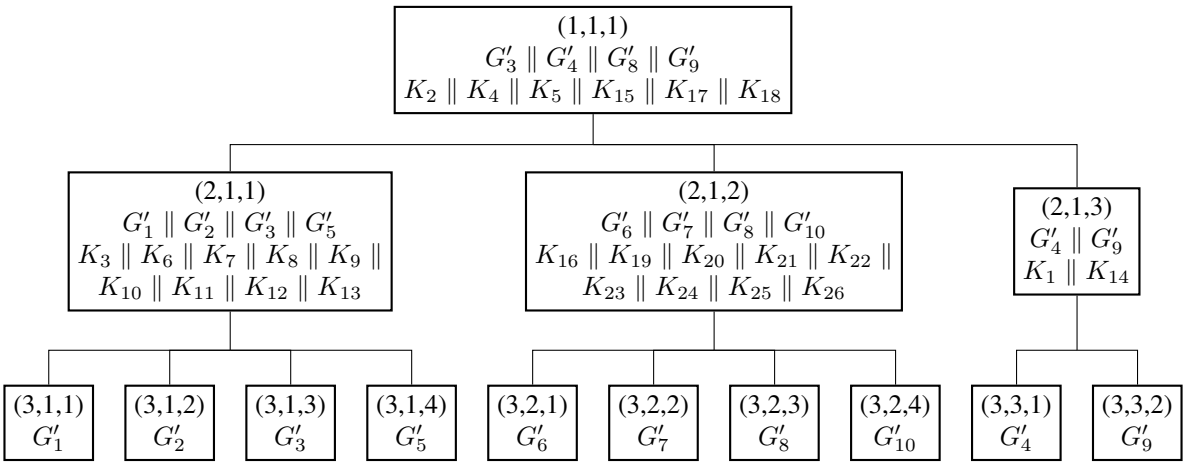

Fig. 4: Tree structure: index set T together with G_n and K_n.

exists at least one requirement which relates the two different subclusters with each other. In Line 8 we identify these requirements by searching the DMM PR. It is possible that one of these particular requirements also relates to other subclusters or even relates to elements inside a particular subcluster. Therefore, we update P and PR by removing all the relationships resulting from the found requirements. When we have finally found all requirements relating the subclusters together at this node, we calculate at Lines 17 and 18 the plant model and requirement model for this node.

Example *Consider the first node* $(1,1,1)$*. In Fig. 3 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* $P(8,3)$*, we search in* PR *for all* $j \in J$ *such that* $PR(8,j) \neq 0 \wedge PR(3,j) \neq 0$*. From Fig. 2, we can see that only* $j = 5$ *and* $j = 18$ *satisfy this condition and are therefore added to* J_n*. Repeating this for* $P(4,8)$ *and* $P(9,3)$ *results finally in* $J_{(1,1,1)} = \{2,4,5,15,17,18\}$*.*

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. 2 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. 4 shows the resulting plant and requirement model at each node. If a node does not show a requirement model* K_n*, none of the original requirements of* $\{K_j\}$ *is placed at this node. Therefore, no supervisory controller is needed.*

By using Algorithm 2 during the creation of the tree structure in Algorithm 1, we can prove the following two theorems, Theorem 2 and 3.

Theorem 2 (Plant model conservation): Consider 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, it follows from Algorithm 2, Line 17, 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', V) 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 will become a set equality, since $\cup_{n\in T,n}$ is leaf node $I_n = I'$. ■

Theorem 3 (Requirement model conservation): Consider Algorithm 1, it holds that $\|_{j\in J} K_j = K = \|_{n\in T} K_n$.

Proof: For each node n visited by Algorithm 1, it follows from Algorithm 2, Line 18, 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 checked by Algorithm 2 when Algorithm 1 is at node n. Therefore, we know for sure 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, V_x) and (A_y, V_y) such that $A_x \neq A_y \wedge a \in A_x \wedge b \in A_y \wedge \exists (A_p, V_p) : (A_x, V_x), (A_y, V_y) \in V_p$. 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, V_p) such that Algorithm 2 checks $P(a,b)$. Therefore, all elements of P are checked. ■

D. Complete algorithm

Algorithm 3 shows the complete set of steps performed to transform a general problem definition into a tree-structured system with at each node a plant and a requirement model. The following theorem states that the result of Algorithm 3 is a valid input for synthesis of a set of supervisors S_s according to [8] and Theorem 1.

Theorem 4 (Valid MLDES tree): Consider a general composed system $\{G_i \mid i \in I\}, \{K_j \mid j \in J\}$ with $G = \|_{i\in I} G_i$, $K = \|_{j\in J} K_j$ and Algorithm 3, the generated output $T, \{G_n \mid n \in T\}$ and $\{K_n \mid n \in T\}$ is a valid input for synthesizing the set $S_s = \{S_n \mid n \in T, S_n \parallel G_n \subseteq K_n\}$

TABLE I: Results of supervisory control synthesis on the simple lock model.

Synthesis architecture	Subsystem	Number of states	Number of transitions
Plant model	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

according to Theorem 1, i.e., $G = \|_{n\in T}\ G_n,\ K = \|_{n\in T}\ K_n$ and $\forall n \in T : K_n \subseteq \Sigma^*_{G_n}$.

Proof: Theorems 2 and 3 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 definition 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 $\Sigma_{K_n} \subseteq \Sigma_{G_n}$. ■

Algorithm 3 TransformToMLDES

Input: $\{G_i \mid i \in I\}, \{K_j \mid j \in J\}$ **Output:** $T, \{G_n \mid n \in T\}, \{K_n \mid n \in T\}$

Ensure: The given problem definition is transformed to a valid MLDES

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 $\Sigma_{G'_i} \cap \Sigma_{K_j} \neq \emptyset, \forall i \in I', j \in J$.
3: Calculate $P = PR \cdot PR^T$.
4: Cluster P, for example with algorithm presented in [14]. Denote the computed multilevel clustering as (I', V).
5: Transform (I', V) to the tree structure including $\{G_n\}, \{K_n\}$ with Algorithm 1 with initial values Transform$C^m$to$T(\emptyset, (I', V), 1, [1], 1, P, PR, \{G'_i\}, \{K_j\}, \emptyset, \emptyset)$.

Example *Fig. 4 shows the result after applying Algorithm 3. Supervisor S_n is synthesized for each node $n \in T$ with a monolithic supervisory control synthesis procedure. Table I shows the number of states and the number of transitions of the supervisors. For a comparison, also the number of states and transitions is shown of the plant model and the supervisory controller obtained with a monolithic synthesis procedure. Since all nodes at level 3 have no requirement, no supervisor is synthesized at these nodes.*

As can be seen, the found 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 set of supervisors is nonconflicting. Therefore, it holds that $\|_{n\in T}\ S_n = S_{mono}$.

PR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	1	1							1	1	1	1				
2				1	1	1	1	1	1	1	1	1	1	1	1	1	
3										1	1				1		
4												1	1				
5														1			
6															1	1	1

Fig. 5: The domain mapping (DMM) matrix PR of the MRI patient support table. Only the nonzero elements are shown for readability.

P	1	2	3	4	5	6
1	7	4	2	2		
2	4	13	3	2	1	2
3	2	3	3			1
4	2	2		2		
5		1			1	
6		2	1			3

P	1	2	3	4	5	6
1	7	4	2	2		
2	4	13	3	2	1	2
3	2	3	3			1
4	2	2		2		
5		1			1	
6		2	1			3

Fig. 6: DSM P for the MRI: left the unclusterd P and right the clustered P_C.

V. Industrial case

In this section, an industrial case is considered. The industrial case is a patient support table of a Magnetic Resonance Imaging (MRI) scanner. For this example, the proposed method is applied such that MLDES supervisory control synthesis can be deployed.

In [18], [19] a supervisory controller of the patient support table of an MRI scanner is designed. The patient support system can be divided into the horizontal axis, the vertical axis and the user interface. The supervisory controller should ensure safe operation of the MRI scanner. The table should move according to the input of the user while ensuring that the table does not move further than the end positions in both horizontal and vertical directions, and the table should not collide with the coil.

The total plant model consists of 32 automata. A detailed description of these plant models can be found in [19], Section 5.3. Please notice that the definition of $\mathbf{hMove}$ should be $\mathbf{hMove} = \{hMoveIn, hMoveOut\}$ and not the definition as presented in Section 5.3 of [19]. In total, 17 requirements are defined and modeled as automata.

First, the system of 32 plant automata is transformed to the most refined product system consisting of only 6 plant automata: VAxis, HAxis, HVNormal, UITumbleSwitch, UIManualButton and UIManualLED.

The DMM of the MRI scanner is shown in Fig. 5. Fig. 6 shows the unclustered and clustered DSM P_C. The clustering is generated with the algorithm presented in [14]

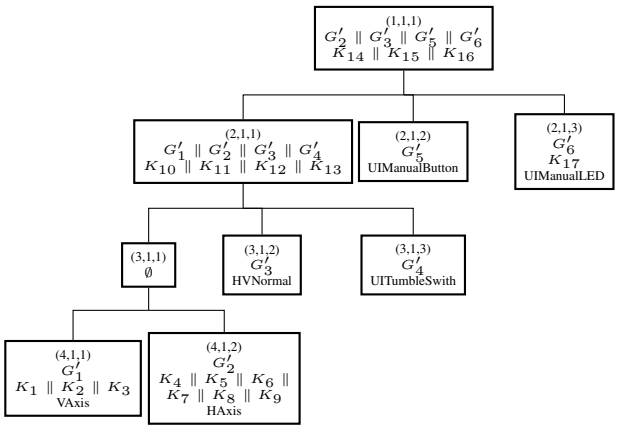

Fig. 7: Tree structure T together with G_n and K_n of the MRI scanner.

TABLE II: Results of supervisory control synthesis on the MRI scanner.

Synthesis architecture	Subsystem	Number of states	Number of transitions
Plant model	G	53.120	464.352
Monolithic supervisor	S_{mono}	30.880	264.456
MLDES supervisors	Sum	6.487	55.642
	$S_{(1,1,1)}$	480	4.528
	$S_{(2,1,1)}$	5.940	50.892
	$S_{(2,1,3)}$	4	12
	$S_{(4,1,1)}$	15	36
	$S_{(4,1,2)}$	48	174

with $\alpha = 1, \beta = 1.2, \gamma = 10$ and $\mu = 1.2$. Note that different parameter values of the clustering algorithm may lead to different, but valid, clusterings.

The final MLDES is given in Fig. 7. The result of synthesizing the set of supervisory controllers of this MLDES is given in Table II. The size of the individual supervisors is significantly smaller than the monolithic supervisor. Furthermore, it holds that this set of supervisors is nonconflicting and thus $\|_{n \in T}\ S_n = S_{\text{mono}}$.

VI. Conclusion and future research

In this paper, a systematic approach is presented to transform a general supervisory control problem definition into a tree-structured MLDES. The key point of this approach is to analyze the interaction between the plant models and the requirement models. After all, supervisory control synthesis algorithms use both plant and requirement models from the complete problem definition. Clustering techniques for DSMs are suitable to find a multilevel clustering for any given supervisory control problem.

Both the explanatory example and the industrial case show a reduction in the number of states and transitions of the synthesized set of supervisors S_s when compared to a monolithic supervisor S_{mono}. The expectation is that, with the proposed method, the number of states and the number of transitions of every supervisor are smaller than those of the monolithic supervisor though there is as of yet no guarantee for this expectation. It is interesting to analyze the relationship between the parameters of the optimization function of the clustering algorithms and the sizes of the synthesized supervisors with respect to the sum of the number of states and the number of transitions.

ACKNOWLEDGMENT

The authors thank prof.em. J.H. van Schuppen for the several fruitful discussions about the control of MLDES and his comments on the 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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] J. Komenda and J. H. van Schuppen, "Coordination control of discrete-event systems," in *WODES*, May 2008, pp. 9–15.

[8] J. Komenda, T. Masopust, and J. H. van Schuppen, "Control of an engineering-structured multilevel discrete-event system," in *WODES*, May 2016, pp. 103–108.

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

[10] 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.

[11] J. Komenda, T. Masopust, and J. H. van Schuppen, "Multilevel Coordination Control of Modular DES," in *CDC*, Florence, Italy, Dec. 2013, pp. 6323–6328.

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

[13] 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, Sept. 2003.

[14] T. Wilschut, P. F. P. Etman, J. E. Rooda, and I. J. B. F. Adan, "Multilevel flow-based markov clustering for design structure matrices," in *IDETC/CIE*. ASME, August 2016.

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

[16] W. M. Wonham, "Supervisory control of discrete-event systems," July 2015. [Online]. Available: http://www.control.toronto.edu/ wonham/

[17] M. H. d. Queiroz and J. E. R. Cury, "Modular control of composed systems," in *ACC*, vol. 6, 2000, pp. 4051–4055.

[18] R. J. M. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. v. Beek, and J. E. Rooda, "Application of Supervisory Control Synthesis to a Patient Support Table of a Magnetic Resonance Imaging Scanner," *IEEE Trans. on Automat. Sci. and Eng.*, vol. 11, no. 1, pp. 20–32, Jan. 2014.

[19] R. J. M. Theunissen, "Supervisory Control in Health Care Systems," Ph.D. dissertation, Eindhoven University of Technology, Eindhoven, 2015. [Online]. Available: http://repository.tue.nl/786117