Supervisory Control of Multilevel Discrete-Event Systems with a Bus Structure

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

***Abstract*— High-tech systems often contain a few components with many dependencies across the system, acting as system-level integrating components. In a hierarchical system decomposition, such so-called bus components tend to be placed in the top node. In supervisory control synthesis, this results in a significant increase of the state space. In this paper, a method is proposed to transform a set of plant models and requirement models into a tree-structured multilevel discrete-event system with a bus structure. After recording the dependencies within the system, the bus and the non-bus plant models are identified and separated. Subsequently, each set of plant models is clustered separately, resulting in a multilevel discrete-event system. Finally, these two systems are merged into a single multilevel discrete-event system where bus plant models are distributed over the multilevel discrete-event system. Experimental results of several models available in the literature are reported to assess the applicability of the proposed method.**

I. Introduction

Supervisory Control Theory (SCT), as introduced by Ramadge-Wonham [1], [2], provides an approach to synthesize supervisory controllers for high-tech systems such that the controlled system behavior is as specified. Recently, SCT has been applied in several case studies including large high-tech systems, such as an advanced driver assistance system [3], a container terminal [4], and a waterway lock [5].

A major drawback of synthesizing supervisory controllers is the computational complexity of the step where the supremal controllable language is calculated. Although the computational 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]. For example, [4] reports that a monolithic supervisor could only be synthesized with the tool Supremica [6], while others fail. Several attempts exploiting different architectures have been proposed to overcome these computational difficulties: modular [7], hierarchical [8], decentralized [9], coordinated [10], and, more recently, multilevel supervisory control synthesis [11].

A problem with several of these supervisory control architectures is that additional information about, for example, the system's structure or controller's structure needs to be provided as input for synthesis. For example, hierarchical supervisory control needs also 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. Recently, an approach is presented in [12] to transform any set of plant models together with the control requirements into the appropriate input for multilevel supervisory control synthesis.

In that paper, the structure embedded in the set of plant models and the set of requirement models is exploited by using Dependency Structure Matrices (DSMs). A DSM is an $N \times N$ matrix capturing the dependencies among N system components. It provides a concise representation for the analysis of the structure of systems in many areas of engineering and research, several industrial examples are provided in [13], [14]. With appropriate analysis techniques, such as clustering, one is able to highlight important aspects in system structures, such as modules of system components.

High-tech systems often contain components with many dependencies across the system, acting as system-level integrating components. These components are called bus components, see [15]. With the approach presented in [12], the bus components are placed high, or even in the top node, in the multilevel system as they are related to many components in the system. In top nodes, this produces large sets of plant models and requirements, resulting in a large supervisory control problem to solve.

The contribution of this paper is a systematic approach to the transformation of a set of finite-automata plant models and a set of requirement models with a bus structure into a tree-structured multilevel discrete-event system (MLDES) for multilevel supervisory control synthesis of [11]. The proposed method tries to place bus plant models as low as possible in the multilevel system, circumventing the creation of large top nodes, by distributing requirements related to bus plant models over the nodes instead of combining them in a single node. After creating a DSM of the system, we identify the bus plant models and separate them from the non-bus plant models. Subsequently, we create an MLDES for the bus and non-bus plant models separately with the method from [12]. Finally, we merge these two MLDESs into a single one, which can be used as input for multilevel supervisory control synthesis. This merging ensures that bus components are located low in the multilevel system and not all together in the top node.

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

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

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

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. 1: Left the unclustered example DSM P and right the clustered DSM P_C revealing two clusters.

presented in Section IV with an illustrative example of a model of a lock. Section V provides experimental results on benchmark testing of the implementation of the proposed method. Section VI concludes the paper.

II. Dependency structure matrix

This section summarizes the concepts and notations of *Dependency Structure Matrices* (also called Design Structure Matrices) used in this paper.

A. Preliminaries

A DSM is a square matrix with the same entities along its axes (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. Fig. 1 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.

A matrix 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 with matrix multiplications is described in [16], [17].

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 [18]. Fig. 1 right shows the clustered example DSM P_C. By reordering the rows and columns of the unclustered DSM P, 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.

A more in-depth introduction to DSM analysis, including notions not used in this paper, is given in [13]. Examples and applications of DSMs can be found in the recent review paper [14].

B. Multilevel Markov clustering

The clustering algorithm of [18] utilizes Markov clustering [19]. 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 [19]. 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 components 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.

Components that have many dependencies across the system, acting as system-level integrating components, are called bus components, see [15]. Non-bus components may or may not be connected with other non-bus components. The heuristic deployed by [18] to identify these bus components uses the node degrees of the weighted graph represented by the DSM. In this heuristic, components are assigned to the bus by repeatedly checking whether the degree of a node is at least a factor γ larger than the median of the degrees of all nodes not yet assigned. When there are no more nodes that satisfies the condition, the remaining nodes are the non-bus nodes. It is advised to have $1.5 \leq \gamma \leq 3.0$, see [18].

Finally, the proposed multilevel clustering of [18] consists of 5 steps: determine the bus nodes, create the bus and non-bus DSM, cluster the bus DSM, cluster the non-bus DSM, reorder the original DSM. The coefficients α, β, μ, and γ are parameters of the algorithm.

III. Supervisory control

This section summarizes the concepts and notations of Supervisory Control Theory used in this paper. A more in-depth introduction to SCT can be found in [20], [21].

A. Preliminaries

Discrete-event systems are typically modeled with finite-state automata (FAs) [20]. An FA is a 5-tuple $G = (L, \Sigma, \rightarrow, l_0, L_m)$ where L is a finite set of locations, Σ a set of events, $\rightarrow \subseteq L \times \Sigma \times L$ is a transition relation, $l_0 \in L$ is an initial location, and $L_m \subseteq L$ is a set of marked locations indicating 'accepting' or 'final' states.

The event set Σ is partitioned into two disjoint sets: the set of controllable events Σ_c and the set of uncontrollable events Σ_{uc}. Controllable events, e.g., switching on an actuator, can be disabled by the supervisor, while uncontrollable events, e.g., changing sensor values, cannot be disabled.

In the framework of supervisory control synthesis [1], 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.

For large-scale systems, a collection of plant models $P_s = \{P_1, P_2, \ldots, P_m\}$ and a collection of requirement models $R_s = \{R_1, R_2, \ldots, R_n\}$ is provided. The collection of plant models and requirements models interact with each other by synchronizing over events by the synchronous product $\|$ [21]. We use the notation Σ_{P_i} and Σ_{R_j} to indicate the alphabet of a plant model P_i or a requirement model R_j, respectively.

The collection of plant models P_s can be transformed into the *most refined product system* P_s' [22] such that each subsystem is independent with respect to any other subsystem, i.e., each subsystem does not share any event with another subsystem. The most refined product system represents the same behavior as the original collection P_s.

B. Monolithic supervisor synthesis

A supervisor controls the behavior of a plant. Supervisory control synthesis [1] provides a method to synthesize a supervisor that adheres to the following control objectives for given plant and requirement models.

- *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 derived from a single plant model and a single requirement model. 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 the requirement model R are obtained by performing the synchronous products.

C. Multilevel discrete-event systems

For the design of controllers, a large-scale system is often decomposed into multiple subsystems. This approach is also applicable to discrete-event systems. A *multilevel discrete-event system* (MLDES) has a tree-based structure, as first proposed in [11]. Each node in the tree has a unique parent (except the top node) and may have children. Each node in the MLDES is interpreted as a subsystem. Therefore, each node consists of a sub-collection of the plant models and a sub-collection of the requirement models. An MLDES is called a coordinated MLDES if all shared events between children are included in their parent.

To control an MLDES with tree structure T, for each node $n \in T$ a supervisor S_n should be constructed such that the synchronous product of all controlled subsystems equals the system controlled by a monolithic supervisor. Supervisor S_n can be obtained by performing a monolithic supervisor synthesis algorithm for the subsystem at node n. It has been shown that such a set of supervisors satisfies the safety and controllability property [11]. A non-blocking check should be performed in the same manner as for modular supervisor synthesis.

D. Structuring MLDES with DSM

The method presented in [12] transforms any set of plant models and requirement models into an MLDES by using DSM-based techniques. The method consists of three steps: recording the dependencies, finding a multilevel clustering, and constructing the MLDES. After obtaining an MLDES, mutilevel synthesis of [11] can be applied.

The first step, recording the dependencies, identifies the relationships between the plant models and the requirement models. As a preprocessing step, the set of plant models is transformed into the most-refined product system. Then, a plant model and a requirement have a relationship if they share an event. A domain mapping matrix PR with the plant models along the rows and the requirement models along the columns is constructed to capture these relationships: $PR(i, j) = 1$ if the plant model on row i has a relationship with the requirement model on column j.

In the second step, finding a multilevel clustering, the obtained DMM PR is transformed into the DSM $P = PR \cdot PR^T$ where PR^T is the transpose matrix of PR. The DSM P is clustered with the multilevel clustering of [18].

Finally, in the third step, constructing the MLDES, the obtained multilevel clustering is analyzed in a top-down manner. Starting from the top node, relationships between the subclusters are identified by searching in the DSM P nonzero entries outside the subclusters. A nonzero entry outside the subclusters means that there exists at least one requirement relating to plant models from both subclusters. Therefore, this requirement should be placed in the top node together with the plant models relating to this requirement. When all nonzero entries in P outside the subclusters have been processed, then the method proceeds by analyzing each subcluster in the same manner until all leaf nodes have been encountered. The resulting tree is an MLDES where each node contains a set of plant models and a set of requirement models.

Fig. 2: The lock at Terneuzen, The Netherlands. Image from https://beeldbank.rws.nl, Rijkswaterstaat.

P_C	3	8	11	1	2	5	6	7	4	9	10
3	17	2	4	4	4	1				2	
8	2	17	4				4	4	2		1
11	4	4	16	1	1		1	1	2	2	
1	4		1	7	2						
2	4		1	2	7						
5	1					1					
6		4	1				7	2			
7		4	1				2	7			
4		2	2						6	2	
9	2		2						2	6	
10		1									1

Fig. 3: The clustered DSM P_C for the simple lock example with a bus.

IV. Bus MLDES

This section presents the proposed method to create an MLDES with a bus structure. The illustrative example of [12] is used to demonstrate the presented approach in this and the next section. This example is here extended with an emergency button to include a clear bus component into the model. The complete model can be found in [23].

Example *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. 2 shows the lock located at Terneuzen.*

After obtaining the most-refined product system, the following subplant models are present in this simplified system:

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*
11) *Emergency button*

On this system, 46 requirements are imposed to guarantee the safe operation of a lock. For example, "if there is no equal water over a gate, then the gate may not be opened."

To analyze a found clustering, we need a formal notion of a multilevel clustering. The following definition from [12] provides one.

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

- If $|A| = 1$, then $(A, A) \in C_A^m$
- If $(A_1, M_1), \ldots, (A_s, M_s)$ with $2 \leq s \leq |A|$ s.t. $\{A_1, \ldots, A_s\}$ is a partition of A and $\forall i, 1 \leq i \leq s : (A_i, M_i) \in C_{A_i}^m$, then $(A, \{(A_i, M_i) \mid 1 \leq i \leq s\}) \in C_A^m$.

A multilevel clustering can be seen as recursively partitioning set A, i.e., set A is partitioned into $\{A_1, \ldots, A_s\}$ where each partition is again partitioned and so on until partitions with a single element are reached. 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 of the set $\{1,2,3\}$.

The first step of [12], recording the dependencies, is still exactly the same for identifying and clustering with bus components. Now, multilevel clustering is applied such that a bus is identified. This will return two multilevel clusterings: the bus clustering (A^B, M^B) and the non-bus clustering (A^{NB}, M^{NB}).

Example *Fig. 3 shows the clustered DSM of the simple lock with a bus. This clustering is obtained with parameters $\alpha = 2$, $\beta = 2.5$, $\mu = 2.5$, and $\gamma = 2.0$. Plant models 3, 8, and 11 are placed in the bus cluster, while the other plant models are in the non-bus cluster.*

Now, each of these two clusters is transformed into an MLDES with Algorithm 1 of [12] given the most-refined product system P_s' and requirements R_s. This results in a tree index set T^B together with a set of plant models $\mathcal{P}^B = \{P_n^B \subseteq P_s' \mid n \in T^B\}$ and a set of requirement models $\mathcal{R}^B = \{R_n^B \subseteq R_s \mid n \in T^B\}$ for the bus cluster, and a tree index set T^{NB} together with a set of plant models $\mathcal{P}^{NB} = \{P_n^{NB} \subseteq P_s' \mid n \in T^{NB}\}$ and a set of requirement models $\mathcal{R}^{NB} = \{R_n^{NB} \subseteq R_s \mid n \in T^{NB}\}$ for the non-bus cluster.

Requirements related to both bus and non-bus components are in both MLDESs. Furthermore, in each node of the bus (resp. non-bus) tree where such a requirement is placed, plant models from the non-bus (resp. bus) clustering are missing. To solve this problem and to assure that each requirement is only located in either the bus or the non-bus MLDES, plant models from the bus MLDES are added to those nodes in the non-bus MLDES that have the above mentioned problem and those requirements are removed from the bus MLDES.

Algorithm 1 shows the procedure to identify the requirements that have a relationship with both bus and non-bus components, and subsequently removes requirements from the bus node and adds the corresponding bus plant models

to the non-bus node.

In the for-loop starting at Line 2, all requirements are checked one by one. In Line 3, it is checked whether requirement j has a relationship with both the bus and the non-bus MLDES. This is achieved by checking whether there exists a plant index assigned to the bus multilevel cluster that has a nonzero entry in the DMM PR and there exists a plant index assigned to the non-bus multilevel clustering that also has a nonzero entry in PR. If that is the case, the algorithm continues in Line 4 by getting the node in the bus MLDES where requirement j is placed in, and in Line 5 by getting the node in the non-bus MLDES where requirement j is placed in. From Theorem 3 in [12], we know that we will find this node, as requirement j was input for both bus and non-bus MLDES creation, and we know that there exists only one such node for both MLDESs. When both nodes have been found, the set $P_{b,j}$ of bus plant models is identified that relate to this requirement j on Line 6. Now we have the required information to remove requirement j from node n^B in the bus MLDES and to add the bus plant models in $P_{b,j}$ to node n^{NB} in the non-bus MLDES.

Algorithm 1 MoveReqFromBusToNonbus

Input: bus clustering (A^B, M^B) with tree index set T^B, the set of plant models $\mathcal{P}^B$, and the set of requirement models $\mathcal{R}^B$, non-bus clustering (A^{NB}, M^{NB}) with tree index set T^{NB}, the set of plant models $\mathcal{P}^{NB}$, and the set of requirement models $\mathcal{R}^{NB}$, the domain mapping matrix PR
Output: adjusted sets $\mathcal{R}^B$ and $\mathcal{P}^{NB}$ where requirements that are related to both bus and non-bus components are removed from $\mathcal{R}^B$ and the corresponding bus plant models are added to $\mathcal{P}^{NB}$

1: Let l be the length of PR
2: **for all** $j \in \{1, \ldots, l\}$ **do**
3: **if** $\exists i^B \in A^B$ with $PR(i^B, j) \neq 0$ and $\exists i^{NB} \in A^{NB}$ with $PR(i^{NB}, j) \neq 0$ **then**
4: Find n^B s.t. $R_j \in R^B_{n^B}$,
5: Find n^{NB} s.t. $R_j \in R^{NB}_{n^{NB}}$
6: $P_{b,j} := \{P_i \mid i \in A^B, PR(i, j) \neq 0\}$
7: $R^{NB}_{n^B_B} := R^{NB}_{n^B} \setminus \{R_j\}$
8: $P^B_{n^{NB}} := P^B_{n^{NB}} \cup P_{b,j}$
9: **end if**
10: **end for**

Finally, the two MLDESs can be joined by including an empty top node as the parent node for the top nodes of the bus MLDES and the non-bus MLDES.

Example *Using the clustered DSM shown in Figure 3, we can identify the bus tree and the non-bus tree. The bus tree consists of a top node an then three leaf nodes. The non-bus tree consists of a top node with five children, of which three consist of two leaf nodes and two are by themselves leaf nodes. The resulting trees are shown in Figure 4, where they are combined by an empty top node.*

For this example, Algorithm 1 adds plant models to nodes $3^{NB}, 4^{NB}, 5^{NB}, 7^{NB}, 8^{NB}, 10^{NB}, 11^{NB}$, *and* 12^{NB} *of the non-bus tree and removes requirements from nodes* $2^B, 3^B$, *and* 4^B *of the bus tree.*

Algorithm 2 is the complete bus MLDES algorithm. It transforms a given set of plant models and requirement models into a bus MLDES with tree-structure T and for each node in T a set of plant models and requirement models.

Algorithm 2 Bus MLDES

Input: set of plant models $P_s = \{P_i \mid i \in I\}$, set of requirement models $R_s = \{R_j \mid j \in J\}$
Output: index set of tree-structure T of bus MLDES, set of plant models $\mathcal{P}_T = \{P_n \subseteq P'_s \mid n \in T\}$, set of requirement models $\mathcal{R}_T = \{R_n \mid n \in T\}$

1: Transform P_s to a most-refined product system $P'_s = \{P'_i \mid i \in I'\}$ with new index set I'.
2: Construct matrix PR such that $PR(i, j) = 1$ iff $\Sigma_{P'_i} \cap \Sigma_{R_j} \neq \emptyset, \forall i \in I', j \in J$.
3: Calculate $P = PR \cdot PR^T$.
4: Perform bus clustering on P, for example with algorithm presented in [18]. Assign the computed bus multilevel clustering to (A^B, M^B) and assign the computed non-bus clustering to (A^{NB}, M^{NB}).
5: Transform (A^B, M^B) to the bus tree structure T^B with $\mathcal{P}^B, \mathcal{R}^B$, and transform (A^{NB}, M^{NB}) to the non-bus tree structure T^{NB} with $\mathcal{P}^{NB}, \mathcal{R}^{NB}$, both by Algorithm 1 of [12].
6: MoveReqFromBusToNonbs$((A^B, M^B), T^B, \mathcal{P}^B, \mathcal{R}^B, (A^{NB}, M^{NB}), T^{NB}, \mathcal{P}^{NB}, \mathcal{R}^{NB}, PR)$.
7: Construct the bus MLDES with $T = \{0\} \cup T^B \cup T^{NB}$, $\mathcal{P}_T = \{\emptyset\} \cup \mathcal{P}^B \cup \mathcal{P}^{NB}$, and $\mathcal{R}_T = \{\emptyset\} \cup \mathcal{R}^B \cup \mathcal{R}^{NB}$.

The following two theorems show that all original plant models and requirement models are somewhere in the constructed bus MLDES, if each requirement has a relation with at least one plant model.

Theorem 1 (Plant model conservation): For an MLDES constructed with Algorithm 2, $\|_{i \in I} P_i = P = \|_{n \in T} P_n$.

Proof: We will follow Algorithm 2 line by line. After Line 1 it follows that $\|_{i \in I} P_i = \|_{i \in I'} P'_i$. Lines 2 and 3 do not alter the most-refined product system. After performing bus clustering in Line 4, we obtain the bus multilevel clustering (A^B, M^B) and non-bus multilevel clustering (A^{NB}, M^{NB}), with $A^B \cup A^{NB} = I'$. Therefore, when combining the results of Line 5 and Theorem 3 of [24], it follows that $\|_{i^B \in A^B} P_{i^B} = \|_{n^B \in T^B} P^B_{n^B}$ and $\|_{i^{NB} \in A^{NB}} P_{i^{NB}} = \|_{n^{NB} \in T^{NB}} P^{NB}_{n^{NB}}$, thus $\|_{i \in I} P_i = (\|_{n^B \in T^B} P^B_{n^B}) \parallel (\|_{n^{NB} \in T^{NB}} P^{NB}_{n^{NB}})$.

Continuing with Line 6 of Algorithm 2 and inspecting Algorithm 1, we observe that plant models present in the bus clustering are added to the non-bus clustering. Therefore, it still holds that $\|_{i \in I} P_i = (\|_{n^B \in T^B} P^B_{n^B}) \parallel (\|_{n^{NB} \in T^{NB}} P^{NB}_{n^{NB}})$. Finally, as Line 7 does not remove plant models during the merge, we can conclude that $\|_{i \in I} P_i = \|_{n \in T} P_n$. ■

Theorem 2 (Requirement model conservation): For an MLDES constructed with Algorithm 2, $\|_{i \in J} R_j = R = \|_{n \in T}$

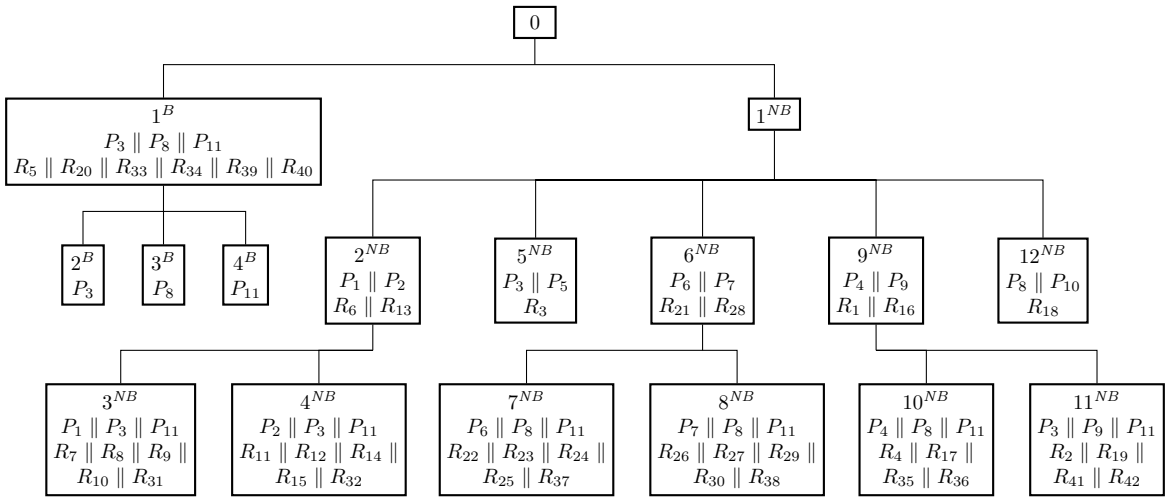

Fig. 4: Tree structure of the bus MLDES: index set T together with G_n and K_n.

R_n if and only if $\forall j \in J \ : \ PR(:,j)^T \cdot PR(:,j) \geq 1$.

Proof: We follow Algorithm 2 line by line. The creation of the most-refined product system in Line 1, the construction of the matrices PR and P, and the multilevel clustering do not alter the requirement set.

Continuing with Line 5, it follows from Theorem 4 of [24] that $\|_{j \in J} \ R_j = (\|_{n^B \in T^B} \ R^B_{n^B}) \parallel (\|_{n^{NB} \in T^{NB}} \ R^{NB}_{n^{NB}})$ if and only if $\forall j \in J \ : \ PR(:,j)^T \cdot PR(:,j) \geq 1$. In Line 6, Algorithm 1 is performed. In Line 7 of Algorithm 1, requirements are removed from the bus multilevel clustering. But this is only done when a requirement has a relationship with both bus and non-bus plant models (Line 3 of Algorithm 1). Therefore, the removed requirements are still present in de non-bus multilevel clustering and thus $\|_{j \in J} \ R_j = (\|_{n^B \in T^B} \ R^B_{n^B}) \parallel (\|_{n^{NB} \in T^{NB}} \ R^{NB}_{n^{NB}})$ if and only if $\forall j \in J \ : \ PR(:,j)^T \cdot PR(:,j) \geq 1$. This concludes the proof, as Line 7 of Algorithm 2 does not remove requirements. ■

Finally, Theorem 3 states that the result of Algorithm 2 is a valid input for synthesis of a set S_s of supervisors according to Theorem IV.5 of [11]. By combining these two theorems, it can be concluded that the presented approach results in the same controlled behavior as a monolithic supervisor would achieve.

Theorem 3 (Valid MLDES tree): Consider a composed system $\{P_i \mid i \in I\}$ and prefix-closed requirements $\{K_j \mid j \in J\}$ with $P = \|_{i \in I} \ P_i$ and $R = \|_{j \in J} \ R_j$, respectively. The output $T, \mathcal{P}_T$, and $\mathcal{R}_T$ generated by Algorithm 2 is an MLDES for synthesis of set $S_s = \{S_n \mid n \in T, S_n \parallel P_n \subseteq R_n\}$ according to Theorem IV.5 of [11] if and only if $\forall j \in J \ : \ PR(:,j)^T \cdot PR(:,j) \geq 1$.

Proof: It suffices to show that $P = \|_{n \in T} \ P_n$, $R = \|_{n \in T} \ R_n$, and $\forall n \in T : R_n \subseteq \Sigma^*_{P_n}$. Theorems 1 and 2 show that $P = \|_{n \in T} \ P_n$ and $R = \|_{n \in T} \ R_n$, respectively. It only remains to be proved that $\forall n \in T : R_n \subseteq \Sigma^*_{P_n}$.

Theorem 5 of [24] applies to the case that requirements only have dependencies either with bus plant models or with non-bus plant models. But in general, there may exist requirements that have dependencies both with bus and non-bus plant models. This issue is resolved by applying Algorithm 1 in Line 6. For those requirements that have dependencies with both bus and non-bus plant models, the nodes in the bus and non-bus MLDES are determined in Lines 4 and 5 of Algorithm 1. Line 6 then collects all bus-plant models related to this requirement. This requirement is removed from node n^B. Doing this for all identified requirements ensures that $R^B_{n^B} \subseteq \Sigma^*_{P_{n^B}}$. The related bus plant models are added to the correct node in the non-bus MLDES (Line 8), which ensures that $R^{NB}_{n^{NB}} \subseteq \Sigma^*_{P_{n^{NB}}}$.

Finally, in Line 7 of Algorithm 2, an artificial top node is added, with $R_0 = \emptyset \subseteq \emptyset = \Sigma^*_\emptyset$. Therefore, we can conclude that in the final MLDES $\forall n \in T : R_n \subseteq \Sigma^*_{P_n}$. ■

TABLE I: Results of supervisory control synthesis on the simple lock model with different supervisor architectures.

Synthesis architecture	Number of states	Number of transitions
Uncontrolled system	165,888	2,248,704
Monolithic supervisor	1,376	8,496
MLDES supervisors	5,690	45,844
Bus MLDES supervisors	253	724

Example *Table I shows the results of synthesizing supervisors for the bus MLDES as shown in Figure 4 with multilevel synthesis of [11], which essentially synthesizes a monolithic supervisor for each node. For this particular example, it is very clear that synthesis for the bus MLDES outperforms the compared synthesis architectures.*

V. Experimental results

In this section, we analyze several models from the literature to assess the applicability of the presented method. We expect to observe the complexity reduction for different examples.

The presented method to transform any problem definition into an MLDES using DSMs has been implemented in the

discrete-event systems tool CIF [25] together with Matlab [26]. It has been tested on several models available in the literature. The models have been selected from [24], where the number of plant models in the most-refined product system is at least 10. The models and results can be found in [23]. A short description of each model is provided below.

central-lock This models the central locking system of a BMW car. The 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 [6].

production-cell In this production cell, metal blanks need to be forged by a press [27]. 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) [3]. 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.

adas* A reformulation of the above adas model where requirements are split into multiple smaller ones when possible, as proposed in [24].

container-terminal A LEGO model of a container terminal system is used to demonstrate model-based engineering [4]. 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 [28]. Products undergo various processing steps in six different workstations: distributing station, handling station, testing station, buffering station, processing station, and sorting station.

lock A waterway lock is used in rivers and canals to raise and lower vessels between different water levels [5]. This model has various subsystems: gates, paddles, culverts, two-lamp traffic lights, and three-lamp traffic lights. An operator interacts with the system through a human-machine interface.

For each model we collect the following metrics: 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 sum of the controlled state-space sizes of the bus multilevel supervisors $bmlcss = \sum_{n \in T_b} mcss_n$, the bus coefficient γ, the number of supervisors of the bus multilevel structure ns, and the calculation time t in seconds.

Table II shows the experimental results for the different models. The models are ordered based on the uncontrolled system state space size. For some models, during the calculation of some metrics an out-of-memory (OoM) error has been encountered as indicated in Table II. A nonblocking check has been performed for those models that have non-prefix closed requirements [21]. 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 experiments showed at first sight mixed results for MLDES with bus. For example, the production-cell and the festo models show an increase in the state-space size of the multilevel supervisors with a bus compared to the multilevel supervisors without bus. All other models in our benchmark show a decrease in the state-space sizes.

Inspection of the individual DSMs, which can be found in [23], shows that both the production-cell and the festo models do not contain clear bus components. In both cases, we force the supervisor to have a certain structure that the system itself does not have. This mismatch is reflected in having a few large nodes in the bus MLDES.

MLDES with a bus is beneficial for several other models, like the central-lock and lock models. The bus MLDES architecture matches these models. For example, the state-space size of the supervisor(s) for the lock model is reduced from $6.0 \cdot 10^{24}$ states of the monolithic supervisor to $7.7 \cdot 10^{6}$ states of the bus MLDES supervisors.

The value of the bus coefficient γ is tuned based on trial and error for each case, as the clustering algorithm of [18] does not provide any heuristics to choose a 'good' value. We optimized the sum of the state-space sizes of the synthesized supervisors, as the state-space size is one of the indications for the complexity of the supervisory control problem. Having a heuristic would increase the applicability of the proposed method for the design of supervisory controllers for large-scale systems, where manual tuning by an engineer is undesired as it may be time consuming.

The experiments show that several supervisors benefit from having a bus MLDES architecture. For the included models, we can conclude based on the unclustered DSM whether it would benefit from bus MLDES.

VI. Conclusion

In this paper, a systematic approach is presented to transform a set of plant models and a set of requirement models into a bus MLDES. System components (i.e., plant models) having many dependencies are separated from the other system components and transformed into an MLDES separately. Having two MLDESs, one for the bus components and one for the non-bus components, these two are merged into a bus MLDES in such a way that bus components are placed in the MLDES as low as possible, to prevent (if possible) a state-space explosion in one of the nodes.

TABLE II: Experimental results for different models

Model	$\|I\|$	$\|J\|$	$\|I'\|$	uss	$mcss$	$mlcss$	$bmlcss$	γ	ns	t
central-lock	74	35	74	$2.6 \cdot 10^5$	OoM	$3.9 \cdot 10^5$	2,108	2.2	15	10.9
production-cell	11	19	10	$3.8 \cdot 10^8$	$1.1 \cdot 10^8$	22,827	67,983	1.5	10	8.6
adas	28	33	27	$3.4 \cdot 10^9$	$2.0 \cdot 10^{10}$	$1.1 \cdot 10^8$	$1.3 \cdot 10^7$	2.0	11	5.4
adas*	28	72	27	$3.4 \cdot 10^9$	$2.0 \cdot 10^{10}$	$5.2 \cdot 10^5$	4,706	2.3	25	5.3
container-terminal	45	35	15	$3.8 \cdot 10^{22}$	OoM	$3.4 \cdot 10^{18}$	$1.3 \cdot 10^{18}$	3.0	13	117
festo	113	211	88	$1.5 \cdot 10^{26}$	$2.2 \cdot 10^{25}$	50,638	$2.6 \cdot 10^5$	3.0	38	10.0
lock	71	198	51	$6.0 \cdot 10^{32}$	$6.0 \cdot 10^{24}$	$3.1 \cdot 10^9$	$7.7 \cdot 10^6$	4.2	34	12.9

Experimental results obtained for a set of models from the literature show that having a bus MLDES architecture can be beneficial. Only models having clear bus components benefit from having the proposed bus MLDES architecture.

Future research could focus on providing guidelines for answering on the question when having a bus MLDES is beneficial without calculating the bus MLDES and synthesizing supervisors. Furthermore, it would be interesting to determine the bus coefficient γ heuristically instead of the current trial-and-error approach. Other optimization criteria may also be interesting to investigate, like the number of supervisors or the depth of the MLDES tree.

ACKNOWLEDGMENT

The authors would like to thank Han Vogel, Maria Angenent and Robert de Roos from Rijkswaterstaat for providing information regarding locks.

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] 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.

[4] M. A. Reniers and J. M. van de Mortel-Fronczak, "An engineering perspective on model-based design of supervisors," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 257–264, May 2018.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] T.-L. Yu, A. A. Yassine, and D. E. Goldberg, "An information theoretic method for developing modular architectures using genetic algorithms," *Research in Engineering Design*, vol. 18, no. 2, pp. 91–109, 2007.

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

[17] 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.

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

[19] 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.

[20] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, 1st ed. Springer, 2018.

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

[22] 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.

[23] M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, and J. E. Rooda, "CIF3 models used for supervisory control of multilevel discrete-event systems with a bus structure." [Online]. Available: https://github.com/magoorden/ECC2019

[24] ——, "Structuring multilevel discrete-event systems with dependency structure matrices," *IEEE Trans. Automat. Contr.*, vol. submitted, 2019.

[25] 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.

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

[27] 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, pp. 1152–1168, Apr. 2009.

[28] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, and J. E. Rooda, "Application of dependency structure matrices and multilevel synthesis to a production line," in *2nd IEEE Conf. on Control Technology and Applications*, 2018, pp. 458–464.