Decomposition-Based Supervisory Control Applied to an Industrial Waterway Lock

Martijn Goorden and Michel Reniers

***Abstract*— The state-space explosion problem limits the use of monolithic supervisory controller synthesis for large discrete-event systems. A solution is to partition the synthesis problem into smaller subproblems. This paper evaluates three decomposition-based multilevel synthesis strategies using the industrial Sluis III waterway lock as a benchmark. All approaches generate correct, globally nonblocking distributed supervisors that are far smaller than the monolithic alternative. Dependency-based clustering yields the most compact supervisors, while specification-structure-based decomposition offers the best balance between efficiency and engineering alignment, demonstrating the effectiveness of decomposition for large-scale supervisory control.**

I. INTRODUCTION

The synthesis of supervisory controllers for complex discrete-event systems (DES) is a cornerstone of modern automation and control engineering. As systems grow in size and complexity, as is observed in manufacturing, transportation, and robotics, the monolithic synthesis approach faces significant challenges. One of them is the state-space explosion problem, which renders monolithic synthesis computationally infeasible for large-scale systems. This motivates the need for decomposition mechanisms that partition the synthesis problem into smaller and tractable subproblems.

Decomposition can be applied at two critical levels: plant models and requirement models. For plant models, decomposition enables modular representation of subsystems, facilitating localized synthesis and verification. This modularity not only improves computational efficiency but also aligns with engineering practices that favor component-based design. Similarly, decomposing requirement models allows constraints to be distributed across subsystems, reducing complexity and enabling incremental verification. However, these benefits come with challenges, for example ensuring that local controllers, when composed, preserve global properties such as safety, nonblocking behavior, and maximal permissiveness.

Over the years, a multitude of decomposition approaches have been developed. [1] categorize architectures into decentralized, distributed, coordination, and hierarchical control. They discuss the formal aspects of each, including coordination needs and communication schemes. In [2] a taxonomy of offline supervisory synthesis is presented, highlighting modular, distributed, and hierarchical approaches.

*Martijn Goorden and Michel Reniers are with Department of Mechanical Engineering, Eindhoven University of Technology, 5612 AZ Eindhoven, the Netherlands m.a.goorden@tue.nl, m.a.reniers@tue.nl

Despite the theoretical foundations of supervisory control theory, its practical application to large-scale systems remains limited due to computational complexity and integration challenges. Existing decomposition approaches often focus on either plant models or requirement specifications in isolation, without fully addressing the interplay between the two. Furthermore, current methods lack systematic guidelines for ensuring that global properties—such as safety, nonblocking, and controllability—are preserved when local controllers are synthesized and composed. This gap hinders the scalability and reliability of supervisory control synthesis in industrial contexts.

By systematically investigating decomposition mechanisms for both plant models and requirement specifications, this paper aims to develop synthesis techniques that are computationally tractable, modular, and robust, thereby advancing the applicability of supervisory control theory to complex real-world systems. To this end, we focus on multilevel synthesis [3], as it performs supervisor synthesis on a tree-based structure of a system, which is a decomposition structure often find in industry.

The core problem is therefore: how can we design and evaluate decomposition mechanisms for both plant models and requirement specifications that enable scalable, modular, and correct supervisory controller synthesis for complex discrete-event systems?

To address this problem, the paper will pursue the following objectives:

- Investigate different decomposition mechanisms. Develop several decomposition mechanisms to translate a given set of plant models and requirements models into a tree-based structure for multilevel synthesis.
- Validate through industrial examples. Apply the developed framework to a representative large-scale system to demonstrate practical feasibility and benefits.
- Evaluate computational efficiency. Assess the proposed decomposition mechanisms using the case study and benchmark them against monolithic synthesis.

In this paper, we study and compare different decompositions for the waterway lock Sluis III. Sluis III serves as a compelling industrial case study since it is a historical lock system whose complex behavior has been effectively captured, controlled, and implemented using formal supervisory synthesis [4]. This example underscores the method's real-world reach from managing enormous state spaces to supporting automated deployment on PLC platforms. This infrastructural object has been chosen because it can be handled by monolithic supervisory controller synthesis using

the tool ESCET/CIF [5], [6]. Therefore, this allows us to compare possible decomposition approaches among each other but also with respect to the benchmark setting of monolithic supervisory controller synthesis.

II. Background

A. Supervisory Controller Synthesis of EFA

Supervisory controller synthesis requires a synthesis specification consisting of a plant specification, describing all possible system behavior, and a requirements specification, describing the allowed behavior.

In ESCET, both are modeled using Extended Finite Automata (EFAs)—finite automata extended with discrete variables. An EFA $(L, V, \Sigma, \rightarrow, L_m, l_0, v_0)$ has locations L, variables V, events Σ, transitions with guards and updates $\rightarrow$, marked locations L_m, and initial location and values for variables l_0 and v_0. A system typically consists of multiple EFAs, combined via synchronous product. ESCET enforces global read, local write semantics: each variable belongs to exactly one EFA but can be read by all. Additionally the following concepts are frequently used:

- For each EFA, a location variable that represent the current location of that EFA.
- Input variables that model externally determined values (e.g., sensor signals).
- Groups that provide hierarchical structuring but do not affect semantics.

Additionally, for requirements convenient additional syntax is available. Requirements may also be written as:

- State invariants, which are Boolean predicates that must always hold (e.g., x $<$ 3). In CIF syntax:

```
requirement x < 3;
```

- State–event invariants, which are conditions (`p`) on when an event (`sigma`) may occur

```
requirement sigma needs p;
requirement p disables sigma;
```

More information on all ESCET language features can be found in 'Language tutorial' and 'Language reference' sections of the online tool documentation through `https://eclipse.dev/escet/cif/documentation.html`.

Monolithic synthesis constructs a supervisor ensuring:

- controllability (only controllable events are disabled);
- nonblockingness;
- safety (all requirements satisfied); and
- maximal permissiveness.

In EFA-based synthesis, guards of plant transitions are strengthened to impose control logic, and consequently the resulting supervisor is provided as an extra EFA that adds guard restrictions for controllable events.

B. Multilevel Synthesis

For large systems, monolithic synthesis may be infeasible. Multilevel synthesis decomposes the specification into a tree, where each node contains a partial synthesis specification (subset of plant and requirement models) [3]. For each node, ESCET performs monolithic synthesis, producing partial supervisors. The synchronous product of these partial supervisors controls the full system.

As discussed in the previous section, ESCET allows engineers to structure their CIF specification using groups, which gives rise to a tree structure, but groups have no semantic meaning. Therefore, from the perspective of performing multilevel synthesis, there is still freedom in choosing which tree structure to use: one could simply follow the structure as provided in the specification, or use heuristics to construct different structures. An example of the latter one is the work of [7], [8], where dependency structure matrices and clustering are used to obtain a multilevel tree that is reflected by the interaction between the plant models and the requirement models. But other decomposition methods are also possible, including manual decomposition.

Then for creating partial synthesis specifications for multilevel synthesis, some constraints need to be taken into account. First, the theory in [3] requires that all events of child nodes are contained in the parent node. If some decomposition does not satisfy this technical requirement, missing events can be added to parent nodes with selfloops. Then, one can argue that synthesis will not add additional guards to events that exclusively appear in models as simple selfloops. Thus, effectively, they could be left out before performing synthesis. In conclusion, this technical requirement can be trivially satisfied in EFA-based synthesis specifications.

The second constraint is that each partial synthesis specification needs to be a valid synthesis specification. This means that events, all discrete variables, and all location variables used in the specification (guards, updates, and requirement invariants) are declared in the same synthesis specification. If not, it is a syntactically invalid CIF specification. If a local partial specification contains some undeclared elements, the specification can be completed as follows:

- Event declarations are added for missing events.
- Input variables are added for missing discrete variables or location variables.

In ESCET, the completion of partial synthesis specifications can be done automatically.

Finally, even when all local supervisors are nonblocking, their combination may become blocking. Therefore, global nonblocking verification is required after synthesis.

III. Case study: Waterway lock Sluis III

The Nieuwe Sluis III (translation: the New Lock III) is a single-chamber waterway lock located on the Wilhelminakanaal near Tilburg, the Netherlands. It is built on an industrial heritage site, as it is located next to a uniquely designed lock from the 1910's that was one of the first with two sequential chambers to handle the considerable elevation differences along the canal. Fig. 1 shows both the old and the new Sluis III. The Nieuwe Sluis III, operationally replacing the original Sluis III, is also unique, as it is outfitted with the world's largest composite lock gates, measuring up to 6.2 m $\times$ 12.5 m. In the remainder of this paper, we will focus on the Nieuwe Sluis III and, for simplicity, call it Sluis III.

Fig. 1. Sluis III, with on the left side the Nieuwe Sluis III and on the right side the original Sluis III. Image from https://beeldbank.rws.nl, Rijkswaterstaat.

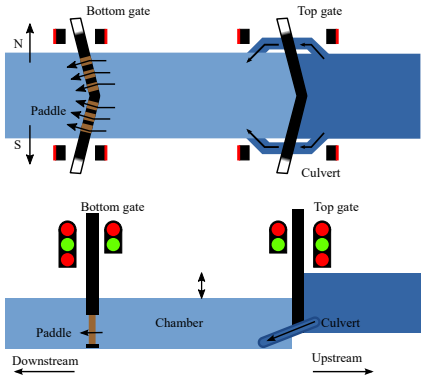

Fig. 2. Schematic representation of Sluis III. Figure from [4].

The schematic representation of Sluis III and its components is shown in Fig. 2. Although the two lock heads of Sluis III are operationally the same, the upstream lock head uses culverts as the water leveling system, while the downstream lock head uses paddles in the gates. As can also be seen from the schematic overview is that on both lock heads the water leveling system and traffic lights are duplicated for redundancy: in case one of them fails, the lock can continue safe operation, minimizing the disruption to traffic while also providing maintenance crew time to fix the problem.

Key control challenges include synchronization of the valve and gate operations across the two lock heads, preventing conflicting water flows or pressure imbalances, ensuring nonblocking behavior so vessels are not stranded mid-process, and maintaining safety and operational reliability under complex dynamics.

Sluis III has been modeled for the purpose of monolithic supervisory controller synthesis in [4], and that model is provided by ESCET as a benchmark model. The plant model used for the treatment of Sluis III in this paper is decomposed into subsystems (e.g., gates, traffic lights). In total, there are 162 plant models involved. In these plant models 25 input variables are used representing a variety of buttons. Safety and functional requirements are expressed using 198 logical state-based expressions.

This models differs from the benchmark model in ESCET in that we removed definitions of automata and requirements and replaced their instantiations with unfolded definitions. This is achieved in ESCET by means of the CIF-to-CIF transformation `elim-comp-def-inst`. We also replaced plant invariants by plant automata (because part of the used functions from ESCET are not available for CIF models with plant invariants). The uncontrolled plant's state space spans approximately 6.0×10^{32} states—indicative of the system's immense complexity.

IV. Different system decompositions for synthesis

In this section, we develop a few different decompositions of the system model with the purpose of synthesizing supervisors for the obtained subsystems (in the next section).

A. Synthesis specification structure decomposition

When closely examining the CIF specification produced for Sluis III (and also for other infrastructural systems, see [9], [10] for example), it can be noted that such specifications are structured. The CIF language offers a grouping mechanism that helps modelers in maintaining and evolving their specifications. In this section, the synthesis specification is decomposed into a number of partial synthesis specifications where the hierarchical structure of the synthesis specification is used to group plants and requirements to form partial synthesis specifications. This results in 17 partial synthesis specifications.

From this group structure a rather straightforward MLDES can be constructed by strictly following the group structures in the synthesis specification. Then, the root node contains a synthesis specification with all plants and requirements that are defined outside the group structure. The 16 groups are then considered the children of the root node. The MLDES that results is given in Figure 3.

B. Clustered most refined product system

In the past decade, the supervisory control group at Eindhoven University of Technology has been struggling with the size of the industrial cases offered by companies in the Brainport region. Therefor, the ESCET / CIF tooling has been extended with a specific multilevel synthesis approach where the decomposition of the synthesis specification into a multilevel DES (see [3], [11], [12]) is determined fully automatically by using a DSM-based clustering approach [7], [8], [13]. In this approach, the plant specification is decomposed into a so-called most refined product system (MRPS). The plant groups in the MRPS are independent of each other, i.e., they do not share events and variables. A DSM is constructed that reflects how often plants are connected by requirements. This DSM is clustered. The hierarchical clustering is then considered an MLDES. Requirements are added to the nodes of the MLDES as described in [7]. The result is a collection of partial synthesis specifications. In the next section, it is explained how this collection of partial synthesis specifications is used to obtain a supervisor for the original input synthesis specification.

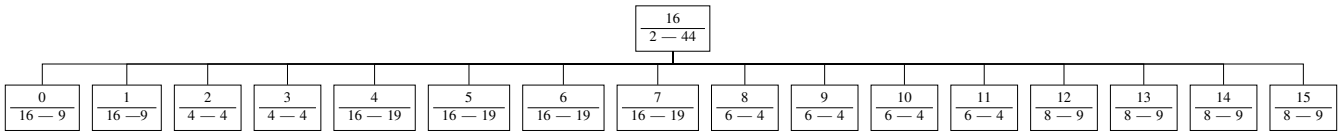

Fig. 3. Structure of the MLDES for clustered synthesis specification decomposition. The numbers in the first row of the nodes indicate the node ID. The numbers in the second row of the nodes indicate the number of plant (left) and requirement (right) automata in the partial synthesis specification of that node.

The MRPS for Sluis III consists of 51 plant groups and 198 requirement groups. After applying DSM-based clustering 37 partial synthesis specifications remain to which synthesis will be applied in the next section. The MLDES that results from clustering and population with requirements is given in Figure 4. In the MLDES, subtrees that only contain empty nodes (i.e., nodes without plants and requirements) are omitted from the MLDES since these nodes do not require synthesis.

The approach sketched here has shown to be of interest for large synthesis specifications, as reported in [14], [15] where this approach has been compared with monolithic synthesis.

A clear benefit of using this approach is that it is implemented in ESCET and can be used out of the box. A disadvantage of this approach may be that, in contrast to the approach in the previous section, the structure that has been provided by the modeler is fully disregarded. This may lead to a bad alignment of the generated supervisors with this structure and consequently also with the system understanding of the user.

Another disadvantage is that the decomposition has to be based on plant groups as provided in the most refined product system. For example in lithography system models that have been provided in literature for the purpose of supervisory controller synthesis [16]–[18] this means that there is only a single plant group containing the complete plant specification, which then does not lead to a non-trivial decomposition at all.

In [7], [8], supervisory control for multilevel discrete event systems with and without a bus structure has been applied to Sluis III. The model that was used is different, and also the clustering parameters used (that are discussed in this chapter) were different from the default settings we used here.

C. Plant decomposition based on system architecture

In this section, we will investigate a decomposition approach that combines the approaches from the previous two sections. Instead of using the plant groups as provided by the most refined product system as the basis for clustering via the requirements, now the partial plant specifications are used to decide on the plant groups. Then, as in the clustered most refined product system decomposition, a clustering algorithm is applied to obtain clusters of plant groups. As before requirements are added to the clusters to obtain a multilevel discrete event system.

While in this case the subdivision into a hierarchical specification structure is based on the physical subdivision of the actual system, the modeler is free to base the specification structure on other criteria.

This way of decomposing a synthesis specification into partial synthesis specifications may result in partial synthesis specifications that refer to entities outside the specification.

Given a decomposition of the plant specification based on its structure into 17 plant groups, we can apply the same DSM-based approach as has been applied in the previous section. The resulting MLDES is given in Figure 5. The number of partial synthesis specifications that results is 20. In section V, synthesis will be applied on these partial synthesis specifications.

V. Synthesis of distributed supervisors

For each of the decompositions introduced in the previous section, as well as for the original synthesis specification, monolithic synthesis is performed using ESCET using default options provided by the tool.

A. Functional correctness

Monolithic synthesis as performed using ESCET, but also using other tools such as Supremica [19], [20], TCT [21], and UltraDES [22], provides a maximally permissive, safe, controllable, nonblocking supervisor for the synthesis specification it is applied to.

For the three non-trivial decompositions provided in the previous section, for each involved partial synthesis specification, a partial supervisor is synthesized using monolithic synthesis. It is possible that two local supervisors (for different subsystems) enforce conditions on controllable events in such a way that the combined behavior (obtained by synchronizing the partial supervisors) is blocking. Therefor, for each of the three decompositions, we have constructed the global supervisor and verified that it is nonblocking.

The size of the distributed supervisors is obtained by adding the state spaces of the partial supervisors obtained from synthesis together, as this relates loosely to the implementation size on PLCs [4]. Note that the cumulative sum of the size of the state spaces of the partial supervisors may not be the best way to compare the computational effort as these are computed and represented symbolically. For the four decompositions discussed, the results are given in Table I.

All three decompositions result in much smaller supervisors than monolithic synthesis. Of these, the clustered most refined product system approach performs far better than the other two.

B. Metrics for comparison of synthesis effort

In the previous section, it is established that the four decompositions result in functionally equivalent supervisors.

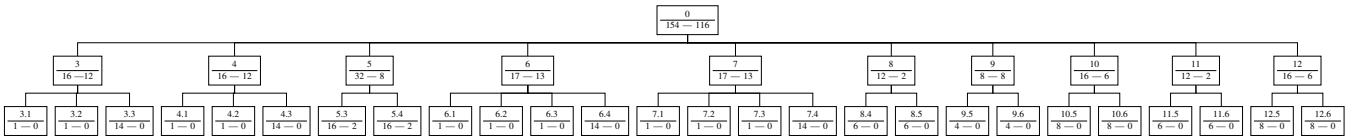

Fig. 4. Structure of the MLDES for clustered most refined product system.

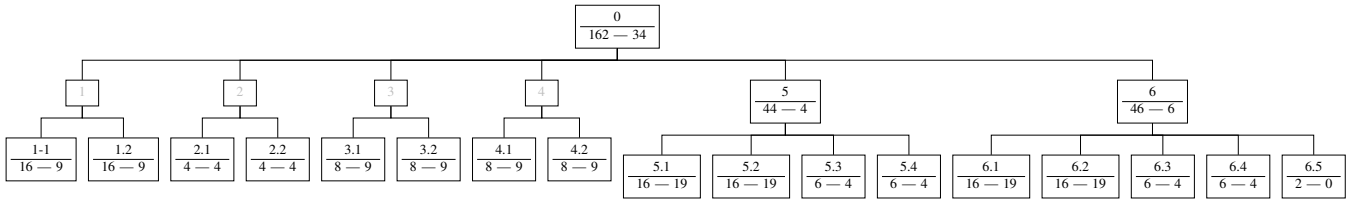

Fig. 5. Structure of the MLDES for clustered plant specification decomposition. The nodes with a light gray number and without information about numbers of plants and requirements are empty specifications.

TABLE I

STATE-SPACE SIZE OF SUPERVISORS FOR THE 4 DIFFERENT DECOMPOSITION APPROACHES.

Decomposition	**Supervisor size** [-]
monolithic (no decomposition)	5.9×10^{24}
synthesis specification structure	1.2×10^{21}
clustered most refined product system	2.8×10^{16}
clustered plant specification structure	2.1×10^{20}

Then, it becomes interesting to explore which of these decompositions resulted in the best synthesis performance.

For comparison of effort, four metrics are used:

1) wall clock time (average over 100 experiments);
2) maximum memory consumption as reported by the synthesis tool (averaged over 100 experiments);
3) maximum number of BDD nodes used during synthesis;
4) number of BDD operations applied during synthesis.

The first two metrics are the most commonly used for comparing the performance of synthesis tools (see for example [20]). A drawback of these is that they typically depend on the computation platform used, and for different computer architectures the performance may appear very different.

When plant and requirement specifications, and consequently their synchronous products, grow large, the computational bottlenecks of supervisory controller synthesis become noticeable: state-space explosion, expensive fixpoint computations, and memory pressure during intermediate constructions. To cope with this, modern supervisory controller synthesis tools, such as ESCET, encode transition relations and supervisors using Binary Decision Diagrams (BDDs) [23], [24]. BDDs are canonical, reduced graph representations of Boolean functions that offer efficient symbolic manipulation and aggressive sharing of substructure [25].

One of the most informative structural metrics for supervisory controller synthesis effort is the maximum number of BDD nodes observed during the synthesis process. This value reflects the maximum symbolic complexity encountered in all intermediate computations, including synchronous product construction, controllability pruning, and nonblocking checks. The maximum node count captures the worst-case memory and computational load.

Complementing structural size, the total number of BDD operations executed during synthesis serves as a direct measure of algorithmic effort. Operations such as apply, relational product, existential quantification, and fixpoint iterations dominate runtime. This metric is particularly useful for profiling and comparing synthesis strategies, as it highlights the cumulative workload imposed.

In [24], it has been shown that the maximum number of BDD nodes and the number of BDD operations are correlated with memory consumption and wall clock time, respectively. Furthermore, these BDD performance metrics are independent of the computational hardware used to run the experiments.

C. Collecting the metrics

The four metrics mentioned in the previous section have been calculated and collected automatically using a ToolDef script, where ToolDef is a platform-independent scripting language as part of ESCET. Below we explain how we collected each metric.

Wall clock time is measured using the Java method `currentTimeMillis` as part of the standard Java `System` class.

The maximum memory consumption can be reported directly by the synthesis tool in ESCET using the `max-memory` option. This measure reports the complete memory usage during synthesis, including the memory used by the ESCET IDE and any other operation running in parallel. Furthermore, Java garbage collection was called at the start of each experiment to ensure that there are no memory leftovers from the previous experiment.

The maximum number of BDD nodes and the number of BDD operations can be requested as statistics from the synthesis tool by using the options `bdd-perf-max-nodes` and `bdd-perf-node-op`, respectively. Since synthesis runs deterministically, no repeated experiments are needed for these performance metrics. All experiments were run on

the same Windows laptop, enabling an as fair as possible comparison between the different decompositions.

D. Comparison of metrics for synthesis effort

In this section, we compare the performance metrics discussed in Section V-B as described in Section V-C for the four decomposed synthesis specifications. The results on the computation effort of synthesis are presented in Table II.

TABLE II
EFFORT METRICS FOR THE 4 DECOMPOSITION APPROACHES.

Decomposition	**Time** [ms]	**Memory** [MB]	**Max. BDD nodes** [-]	**BDD ops.** [-]	**Supervisor size** [-]
monolithic	178	124	6510	216723	5.9×10^{24}
synthesis spec.	236	247	563	7074	1.2×10^{21}
clustered MRPS	1125	343	3829	171056	2.8×10^{16}
clustered plant spec.	371	269	5054	174260	2.1×10^{20}

All three non-trivial decompositions require more time and memory than monolithic synthesis. This may be caused by the time that the tool needs for functions that are not directly related to synthesis computations such as writing output files. When synthesis is applied multiple times, these time-consuming components are repeated also multiple times.

Note that from the non-trivial decompositions the one that is based on the synthesis specification structure requires much less effort in both number of BDD nodes and number of BDD operations. The smallest resulting supervisor (measured in summed state space sizes of partial supervisors) is obtained with the clustered most refined product system.

VI. CONCLUDING REMARKS

This paper demonstrates that decomposition-based multi-level synthesis is a viable strategy for addressing the scalability challenges inherent in supervisory controller synthesis for large-scale discrete-event systems. Using the Sluis III waterway lock as a case study, we evaluated three decomposition approaches against monolithic synthesis. All approaches produced functionally correct supervisors that preserved safety, controllability, nonblockingness, and maximal permissiveness.

Our experiments reveal that decomposition significantly reduces supervisor size. However, this benefit comes at the cost of increased synthesis time and memory usage. Among the tested methods, DSM-based clustering of the most refined product system achieved the smallest supervisors, while specification structure decomposition offered the best trade-off between computational effort and alignment with engineering practices.

REFERENCES

[1] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*. Springer, 2019.

[2] W. Fokkink and M. Goorden, "Offline supervisory control synthesis: taxonomy and recent developments," *Discrete Event Dynamic Systems*, vol. 34, no. 4, pp. 605–657, 2024.

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

[4] F. Reijnen, M. Goorden, J. M. van de Mortel-Fronczak, and J. Rooda, "Supervisory control synthesis for a waterway lock," in *CCTA*. IEEE, 2017, pp. 1562 – 1568.

[5] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. T. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers, "CIF 3: Model-based engineering of supervisory controllers," in *TACAS*. Springer, 2014, pp. 575–580.

[6] W. J. Fokkink, M. A. Goorden, D. Hendriks, D. A. van Beek, A. T. Hofkamp, F. F. H. Reijnen, L. F. P. Etman, L. Moormann, J. M. van de Mortel-Fronczak, M. A. Reniers, J. E. Rooda, L. J. van der Sanden, R. R. H. Schiffelers, S. B. Thuijsman, J. J. Verbakel, and J. A. Vogel, "Eclipse ESCET™: The Eclipse supervisory control engineering toolkit," in *TACAS*. Springer, 2023, pp. 44–52.

[7] M. Goorden, J. v. d. Mortel-Fronczak, M. Reniers, W. Fokkink, and J. Rooda, "Structuring multilevel discrete-event systems with dependence structure matrices," *IEEE Transactions on Automatic Control*, vol. 65, no. 4, pp. 1625–1639, 2020.

[8] M. A. Goorden, C. Dingemans, M. A. Reniers, J. M. van de Mortel-Fronczak, W. J. Fokkink, and J. E. Rooda, "Supervisory control of multilevel discrete-event systems with a bus structure," in *ECC*. IEEE, 2019, pp. 3204–3211.

[9] F. F. H. Reijnen, M. A. Reniers, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Structured synthesis of fault-tolerant supervisory controllers," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 894–901, 2018.

[10] M. M. Baubekova, K. J. van Eldik, J. M. van de Mortel-Fronczak, W. J. Fokkink, and J. E. Rooda, "SBE configurator: A model generation tool for synthesis of ship lock supervisors," *IFAC-PapersOnLine*, vol. 58, no. 1, pp. 288–293, 2024.

[11] J. Komenda, T. Masopust, and J. H. van Schuppen, "Multilevel coordination control of modular DES," in *CDC*. IEEE, 2013, pp. 6323–6328.

[12] ——, "Maximal permissiveness of modular supervisory control via multilevel structuring," *IFAC-PapersOnLine*, vol. 53, pp. 2116–2121, 2020.

[13] 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 *CCTA*. IEEE, 2018, pp. 458–464.

[14] M. M. Baubekova, M. A. Goorden, M. A. Reniers, J. M. van de Mortel-Fronczak, J. E. Rooda, and W. J. Fokkink, "Supervisor synthesis for multilevel DES with local buses," *IFAC-PapersOnLine*, 2026, to be published. [Online]. Available: https://arxiv.org/abs/2512.01101

[15] D. Hendriks, M. Reniers, W. Fokkink, and W. Oortwijn, "Overview and performance evaluation of supervisory controller synthesis with Eclipse ESCET v4.0," 2025. [Online]. Available: https://arxiv.org/abs/2511.04370

[16] J. W. M. Michels, "Supervisory control of ASML wafer scanners," Master's thesis, Eindhoven University of Technology, Department of Mechanical Engineering, 2012.

[17] 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," in *MODELS*. IEEE, 2015, pp. 416–425.

[18] A. E. Hajj Hassan, "Modelling and supervisory control synthesis for engine manufacturing plants," Master's thesis, Eindhoven University of Technology, Department of Mechanical Engineering, 2025.

[19] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica – an integrated environment for verification, synthesis and simulation of discrete event systems," Tech. Rep., 2006. [Online]. Available: http://sourceforge.net/projects/fuber

[20] 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, 2017.

[21] L. Feng and W. M. Wonham, "TCT: A computation tool for supervisory control synthesis," in *WODES*. IEEE, 2006, pp. 388–389.

[22] L. V. Alves, L. R. Martins, and P. N. Pena, "UltraDES – a library for modeling, analysis and control of discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5831–5836, 2017.

[23] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic computation of reduced guards in supervisory control," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.

[24] S. Thuijsman, D. Hendriks, and M. Reniers, "Reducing the computational effort of symbolic supervisor synthesis," *Discrete Event Dynamic Systems*, vol. 34, p. 689–732, 2024.

[25] R. E. Bryant, "Symbolic boolean manipulation with ordered binary decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, Sept. 1992.