The Road Ahead for Supervisor Synthesis

M.A. Goorden, L. Moormann, F.F.H. Reijnen, J.J. Verbakel, D.A. van Beek, A.T. Hofkamp, J.M. van de Mortel-Fronczak, M.A. Reniers, W.J. Fokkink, J.E. Rooda, and L.F.P. Etman

Eindhoven University of Technology, The Netherlands

Abstract. This paper reports on recent research advances in supervisor synthesis, as well as industrial applications and future research challenges, especially in the context of a research project funded by Rijkswaterstaat, responsible for the construction and maintenance of infrastructure in the Netherlands.

1 Introduction

Rijkswaterstaat, as part of the Dutch Ministry of Infrastructure and Water Management, is responsible for the design, construction, management, and maintenance of the main infrastructure in the Netherlands. This includes roads, bridges, tunnels, and waterway locks. In the coming decades, many such systems will be renovated or replaced, as they reach their end of life cycle or have capacity problems. In the past, they were engineered, built, and maintained on an individual project basis, resulting in a large variety of solutions to the same engineering problem. Rijkswaterstaat is seeking methods for modularization and standardization to increase quality and evolvability, decrease life-cycle costs, and enable so-called smart mobility. More and more functionality of infrastructural systems is being automated. The quality of supervisory controllers for such systems has a significant impact on their availability and reliability.

Supervisory control theory is a model-based methodology for designing supervisory controllers. Supervisor synthesis allows to automatically calculate a supervisor. From the supervisor, a supervisory controller can be derived. Taking the uncontrolled behavior of the system components as starting point, an engineer needs to formulate system requirements that rule out all undesired behavior, while allowing desired behavior. The supervisor is then synthesized automatically from the requirements together with the unrestricted system behavior. This is achieved by blocking controllable (output) events, such as starting a motor, as opposed to uncontrollable (input) events, such as sensor reports, over which the supervisor exerts no control. This automatic synthesis may at first sound as some kind of wizardry, but in essence the underlying idea is simple. Let us take the example of a bridge. The software units that drive the different components of the bridge, such as barriers, traffic lights, and the motors to lift the bridge decks, are all supposed to be operational. The supervisor is only required to block illegal activities, such as opening the bridge while the barriers have not

yet closed. Such illegal activities are pre-determined from the requirements and then made unreachable by blocking as few controllable events as possible.

Rijkswaterstaat, in an effort to regulate and unify the system designs of its diverse contractors, has developed a strict regime of structuring and formulating requirements, which form the starting point and basis for system implementations. Additionally, infrastructural systems under their responsibility are mostly discrete-event systems with a clear distinction between controllable and uncontrollable events. This makes supervisor synthesis a suitable engineering method for these applications, which can contribute to the aim of Rijkswaterstaat to increase the quality and evolvability of its supervisory controllers.

In the past, the applicability of supervisor synthesis in real-life systems engineering was often limited due to the heavy demand on computational power, both in time and memory. Recent research developments have helped to make this methodology applicable to large real-life infrastructural systems. This paper discusses some of these developments as well as several case studies in the context of the Rijkswaterstaat project, which all use the CIF 3 toolset. These case studies concern the development of supervisory controllers for bridges, tunnels, waterway locks, and roadside systems. We also discuss future research challenges to further strengthen the applicability of supervisor synthesis.

2 Supervisory control theory

The task of a supervisory controller is to control the different components and subsystems such that the entire system, called the plant, behaves as intended. A supervisory controller is part of a control stack with different layers, see Fig. 1 below. A human operator sends instructions to the supervisory controller via a graphical user interface (GUI), such as to raise a sluice gate inside a waterway lock. The GUI moreover displays sensor data from the plant, such as the height of the water inside the lock. At the plant, actuators drive the mechanical components of the plant, while sensors pick up information regarding the plant. A supervisory controller constitutes a layer between the GUI and the actuators and sensors. Signals to actuators can be blocked by the supervisory controller, i.e., are controllable, if they would lead to violation of a requirement on the plant's behavior. Signals from sensors are uncontrollable for the supervisory controller and are always allowed to occur.

Supervisory control theory, initiated by Ramadge and Wonham [26], automatically synthesizes a supervisor from formal specifications of (1) the plant components and (2) the requirements imposed on the behavior of the plant. This supervisor can then be used to derive a supervisory controller, as shown in [4, 15]. Here we will focus on formal specifications expressed by so-called extended finite automata (EFAs), consisting of finite automata embellished with discrete variables. Guards and updates can be added to transitions that use such variables. EFAs have the same expressive power as finite automata [36], but tend to provide a much more concise representation of the modeled behavior.

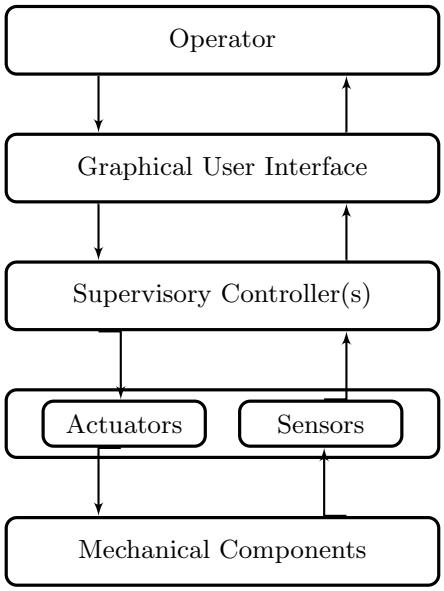

Fig. 1. Schematic view of a control system structure of an infrastructural system

The components of the plant are specified as EFAs and put in a synchronous product, meaning that a certain event, with a particular action name, can only be performed if each component that contains this event is in a state where it can perform this event. This synchronous product specifies the behavior of the entire plant. The requirements that are imposed on the plant behavior are also specified in the form of EFAs. To achieve a supervisor, the requirements are added to the synchronous product. As a result, events that can be performed by the plant may be blocked because they are not allowed by a requirement.

The requirements concern safety properties, making sure that no bad things will ever happen. To ensure liveness, meaning that eventually something good will happen, some states in the EFAs of plant components can be marked by the modeller. A marked state in the synchronous product means that all individual plant components are in a marked state, expressing that the plant is in a steady position, e.g., the bridge deck is closed, the barriers are open, and the signals are green. It must be guaranteed that the plant can always reach a marked state. Therefore, (events leading to) states that violate this property are blocked.

In summary, the synthesized supervisor is (1) *safe*, meaning that all requirements are satisfied, (2) *controllable*, meaning that it only blocks controllable events, and (3) *nonblocking*, meaning that the plant can always reach a marked state. Moreover, it is (4) *maximally permissive*, meaning that the supervisor imposes the minimum restrictions to enforce the first three properties.

A nice introduction to supervisory control theory is offered in [2].

3 Techniques to enhance supervisor synthesis

This section discusses different techniques that help to scale supervisor synthesis to large-scale industrial case studies, how additional features such as fault-tolerance and counterexample generation can be included in the synthesis process, and that implementation code for a programmable logic controller (PLC) can be generated from a supervisor.

3.1 Multilevel synthesis through design structure matrices

The number of states of a plant grows exponentially with its number of components. For example, in [29], a supervisor was synthesized for a relatively small waterway lock, called Lock III, consisting of a single chamber in the Wilhelmina canal at the city of Tilburg. The combination of 4 incoming and 4 outgoing traffic lights, 2 gate paddles, 2 culverts, 4 gates, 2 equal-water sensors, and 25 GUI buttons gives rise to an uncontrolled plant that has $6.0 \cdot 10^{32}$ states. In total systems engineers formulated 142 requirements for this plant. Its controlled state space, monitored by the maximally permissive supervisor, still consists of $5.9 \cdot 10^{24}$ states. A key challenge in performing supervisor synthesis for large real-life infrastructural systems is to cope with this inherent state space explosion problem. The experiments on Lock III reported in this section were redone especially for the current paper, to achieve consistent numbers on sizes of state spaces for the different optimizations. All models and results can be found on a Git repository[1].

In *modular* supervisor synthesis [41], for each individual requirement a different "local" supervisor is synthesized. An event is enabled by the overall supervisor if all local supervisors enable it. An important optimization with regard to modular synthesis, proposed in [25], is to consider for each requirement only those plant components that have at least one event in common with this requirement. This can lead to a drastic reduction in the sizes of the local supervisors. *Multilevel* synthesis [10] reduces the large number of local supervisors generated in modular synthesis by grouping together subsets of components and their related requirements and generating a local supervisor for each such cluster. A key underlying idea is to keep track which pairs of distinct plant components have a requirement in common, in the sense that the requirement is related to both components. This is documented in a so-called design structure matrix (DSM), with one row as well as one column for every plant component. A field in the matrix is colored gray if the components associated to its row and to its column have a requirement in common. An algorithm from [40] determines a clustering of components such that gray fields in the matrix are placed as close as possible to the diagonal. In Fig. 2, a DSM for Lock III is depicted. The clustering algorithm in this case determines clusters which can be mapped back to physical entities: gates, traffic lights, water leveling systems, and the stop and emergency stop.

[1] https://github.com/magoorden/SETTA2020

Synthesis is performed for the plant components in each cluster separately, together with all related requirements for this cluster. Synthesis continues in a hierarchical fashion: next it is performed for related clusters of clusters, et cetera, until finally the overall supervisor for the entire plant is computed. Multilevel synthesis for Lock III produces local supervisors that in total require $1.8 \cdot 10^{23}$ states, as compared to the aforementioned $6.0 \cdot 10^{24}$ states for a monolithic supervisor.

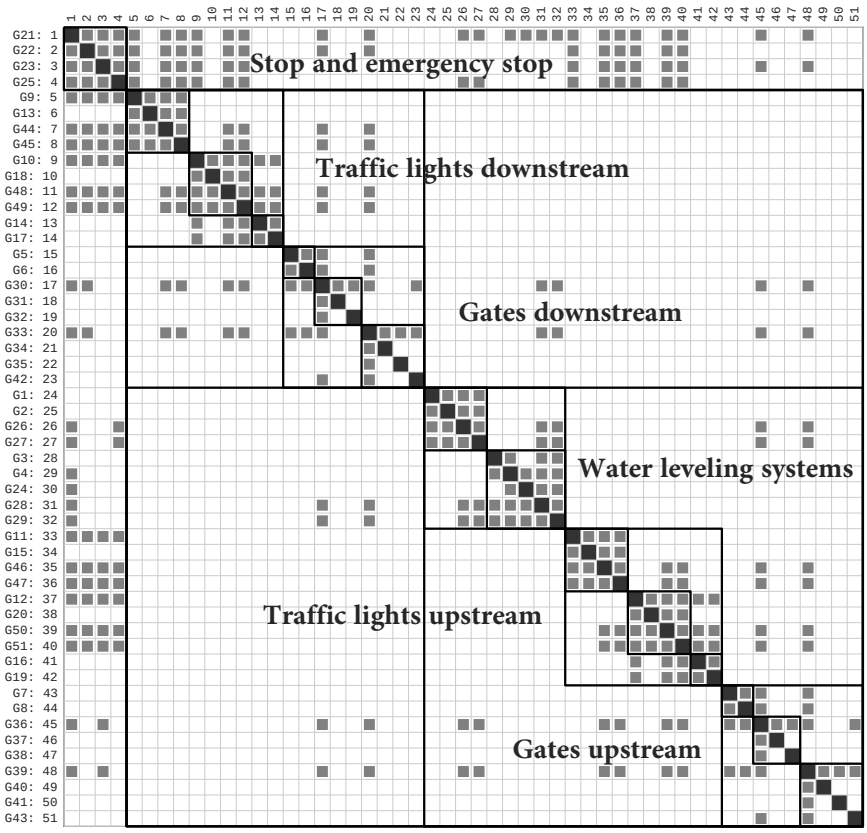

Fig. 2. Clustered DSM for Lock III

It turns out to be very important for the efficiency of multilevel synthesis to split a requirement into smaller requirements when possible [8]. The reason is that this gives rise to fewer relations between the different plant components, so that smaller clusters of components can be synthesized at the start. For example, one requirement of Lock III states that it is unsafe to open a gate when:

– the water-leveling system at the other side is not closed, or
– the gate at the other side is not closed, or
– there is no equal water over the gate.

This requirement can be divided into 13 smaller requirements: (1) split the 3 disjuncts; (2) disentangle references to multiple system states (e.g., "not closed"); and (3) disentangle references to multiple plant components (e.g., gates consist of multiple components). Overall, in [8], the 142 requirements for Lock III are split into 358 requirements. As a result, the number of states required for the local supervisors drops to $4.5 \cdot 10^{19}$ states.

Similar to splitting requirements, multilevel synthesis is also strengthened by splitting plant components into smaller components. In [9], the 35 components are split into 51 smaller components, yielding a reduction of the local supervisors to $1.5 \cdot 10^{19}$ states with respect to the original model.

Combining splitting of requirements and splitting of plant components improves the efficiency of multilevel synthesis significantly. Having 51 plant components and 358 requirements results in the number of states required for the local supervisors to drop spectacularly to $2.9 \cdot 10^{10}$ states.

Bus structures inside a plant, which have requirements in common with most of the plant components, turn out to cause a bottleneck for multilevel synthesis. The stop and emergency stop for Lock III form an example of such a bus structure. Bus components are usually considered only at the final phase of multilevel synthesis, when the overall supervisor is computed. This tends to produce a relatively large supervisor at this final phase. Therefore, in [5] it is proposed to treat bus components separately in multilevel synthesis. At each synthesis phase, concerning certain non-bus components and the requirements they have in common, the bus components that are related to at least one of these requirements are included, thus avoiding the need to treat the bus structure separately at the very end. For instance, for Lock III, treating bus components in this way produces local supervisors that in total require only $7.7 \cdot 10^8$ states.

Next to guiding the hierarchical structuring of clusters of components within multilevel synthesis, DSMs also help in capturing modeling errors of the actual plant. For example, let us consider only the stop and emergency stop of the DSM for the original model of Lock III, detailed in Fig. 3

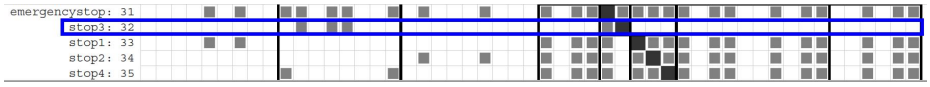

Fig. 3. Snippet of the clustered DSM for Lock III indicating a modeling error

Since the functionality of `stop1`, `stop2`, `stop3`, and `stop4` is similar, one would expect that the rows for these four components in the DSM would be similar. However, the highlighted row for `stop3` is clearly distinct from the rows for the other three components. This observation led to the detection of a small but

serious copy/paste error in several requirements formulated by system engineers, where the event name `stop4` should actually be `stop3`. Based on a series of industrial case studies, in [7] three modeling guidelines are formulated in relation to DSMs. (1) *Similar components should have similar relationships in a DSM.* (2) *A DSM should not contain an empty row or column.* (3) *A DSM should not have independent clusters.* The first two guidelines led to the detection of modeling flaws in plant components and requirements, while the third guideline led to the detection of missing requirements.

In this section Lock III was used as a running example. Another illuminating example of the strength of combining multilevel synthesis with DSMs can be found in [28], where this approach is applied to a small-scale production line, from modeling the plant and its requirements up to the automatic generation of controller code for implementation.

3.2 Additional features for supervisor synthesis

We discuss several advancements and case studies in supervisor synthesis that add features needed for industrial applications and pave the way to apply supervisory control theory from formal plant models and requirements up to the generation of efficient PLC code for a controller that implements the supervisor.

Nonblockingness check for modular synthesis The Achilles' heel of modular supervisor synthesis is that a collection of nonblocking local supervisors may still induce a blocking overall supervisor, meaning that the latter supervisor contains states from which no marked state is reachable. In [18], a method is introduced to efficiently check for nonblockingness, by first performing a string of blockingness-preserving abstraction steps on the local supervisors, resulting in a simplified monolithic supervisor, which is blocking if and only if the overall supervisor induced by the local supervisors in blocking. In [11], reversals of those abstraction steps are defined, so that a blocking simplified supervisor can be used to transform the corresponding overall supervisor into a nonblocking one (which is safe, controllable, and maximally permissive).

Achieving nonblockingness through a dependency graph In [6], properties are formulated on plant components and requirements which ensure that the resulting supervisor is nonblocking. Roughly these are: (1) plant automata are strongly connected, (2) different plant automata do not have events in common, (3) requirements express in which states some plant automata must be to perform some controllable event, and (4) the plant automata mentioned in a requirement only consist of uncontrollable events. Properties (1) and (4) ensure that plant components can always be brought in a state needed to satisfy a certain requirement, as expressed in (3). And (2) makes sure a plant component can be brought in a desired state while the other plant components remain stationary. In modular synthesis this allows to determine clusters where nonblockingness is obtained for free.

In practice, a stumbling block for this method tends to be violation of property (4). Therefore, in [12], a dependency graph is introduced, in which the nodes are plant components and there is an edge from a node P to a node Q if a requirement expresses a condition on a controllable event that occurs in plant P, while the requirement requires plant Q to be in a certain state. Plant components that do not exhibit an infinite path in this graph, do not give rise to blocking behavior. For example, the following dependency graph in Fig. 4 is generated for the model of Lock III.

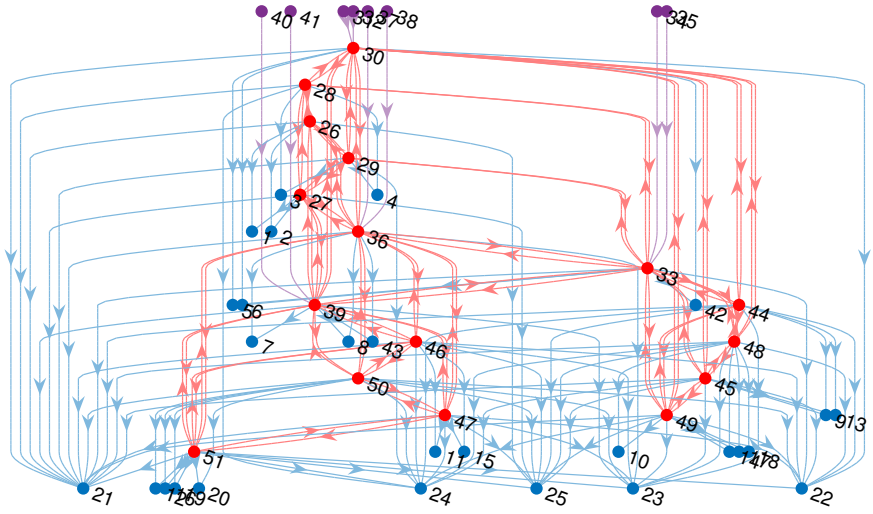

Fig. 4. The dependency graph for Lock III

The strongly connected components in this graph, drawn in red, require a nonblockingness check, as well as nodes from which a path exists to a strongly connected component, drawn in purple. In total 27 plant components and related requirements of Lock III can thus be omitted from the nonblockingness check.

Symmetry reduction Validation of supervisory controllers is an important step to guarantee correct and safe system behavior. In [20], a method is proposed to significantly reduce the validation effort by exploiting that often different plant components are equivalent in behavior, and likewise for different requirements, modulo some renamings of events. This symmetry reduction was essential in a case study on tunnel systems. A supervisor for subsystems such as lighting, ventilation, and emergency detection in the First Heinenoordtunnel, a roadway tunnel with two traffic tubes located south of the city Rotterdam, could be validated without exploiting symmetry [21]. However, for the King Willem-Alexandertunnel at the city of Maastricht, the first two-layered roadway tunnel

in the Netherlands, with two traffic tubes at each layer, the supervisor, became so large that it could only be validated after applying symmetry reduction [20]. This research line has also been combined successfully with the work on dependency graphs in [22].

Counterexample generation The generated supervisor may not be according to the expectations of the user, typically because it does not allow any plant behavior. In such cases it is desirable to obtain feedback on the synthesis process, for instance to detect a modeling error that causes aberrations in the generated supervisor. In [38], a collection of deduction rules are presented that allow to derive reasons for the absence of a state in a supervised system and provide feedback to the user. An adaptation of a standard synthesis algorithm is provided to automatically obtain a cause for each state that is omitted from a plant during synthesis.

Synthesis of fault-tolerant supervisors A fault-tolerant supervisor [19] enables the plant to continue its intended operations, possibly with degraded service, when some of its components fail or a fault occurs, such as a broken wire, defect sensor, or blocking actuator. In [24], after detection of a fault, an alternative supervisor is activated, designed specifically for this type of fault. An alternative approach is followed in [34], where the same supervisor controls the plant before and after the occurrence of faults. Instead, the models of the plant components and the requirements are divided into fault-free and post-fault behavior, where the latter depends on the type of fault that is diagnosed. This approach was successfully applied to synthesize a fault-tolerant supervisor for the Algera bridge over the river IJssel (see Fig. 5), which consists of two vehicle lanes, a lane for cyclists, and a lane for pedestrians. This supervisor allows the bridge to continue its operation when only one out of a pair of stop signs breaks down, and restricted operation when one of a pair of barriers remains stuck in the closed position. Furthermore, it takes appropriate action if the bridge deck opens erroneously.

Continuous-time simulation A hybrid plant model is obtained by extending the discrete-event model used for synthesis with continuous behavior. This is modeled by continuous variables that change their values at the passing of time, defined by a differential equation that can depend, for example, on the current location of a plant component. Additionally, continuous variables may change their values during an event. In [30], this approach was used to simulate a supervisor for the Algera complex, consisting of the aforementioned bridge together with a waterway lock and two storm surge barriers.

Hardware-in-the-loop simulation While model simulation is a powerful tool for validation, it offers only a partial analysis. Aspects related to the execution of the supervisory controller on the hardware and communication with subsystems, such as GUIs, cannot be validated with model simulation. To bridge the gap between model simulation and realization, hardware-in-the-loop simulation can be performed after model simulation and before implementation, meaning that the

Fig. 5. The Algera complex consisting of a lock, a bascule bridge, and two storm surge barriers [https://beeldbank.rws.nl, Rijkswaterstaat / Joop van Houdt].

supervisory controller realization and its subsystems are connected to a model of the plant. In [35], an engineering method is proposed that combines supervisor synthesis with hardware-in-the-loop simulation. Models created for synthesis are refined and re-used to obtain models for model simulation and hardware-in-the-loop simulation. This method was applied successfully to the Prinses Marijke complex in the Amsterdam-Rhine canal, consisting of two waterway locks and a floodgate as protection from the river Lek.

Confluence and finite response While nonblockingness of a supervisor guarantees that always a marked state of the plant is reachable, it does not guarantee that a deterministic supervisory controller implemented on the basis of such a supervisor ever reaches a marked state [4]. In the transformation from a synthesized supervisor to an actual supervisory controller that drives the mechanical plant by enforcing events, it is important to know whether the supervisor exhibits *finite response* and *confluence* [3]. A state of the plane is called stable if the supervisory controller cannot enforce any event, typically because all events leaving this state are uncontrollable. Usually the marked states of a plant are stable. Finite response guarantees that each execution trace allowed by the supervisor eventually returns to a stable state, so that clearly this property is also satisfied by the supervisory controller implemented on the basis of this supervisor. This ensures that a supervisory controller cannot continuously enforce events. Confluence of a supervisor guarantees that each enforcement of events by the

supervisory controller leads back to the same stable state. In [31], building on earlier work in [15], sufficient criteria are given to guarantee that a synthesized supervisor exhibits finite response, and also sufficient criteria to guarantee that it is confluent.

Generation of PLC code For the implementation of supervisory controllers, often a programmable logic controller is used. In [37], PLC code is generated automatically from a model of the supervisor. In recent follow-up work, the structure of generated PLC code has been improved and optimization techniques have been used to reduce its variability in execution times.

In industry, supervisory controllers need to adhere to strict safety standards. To achieve these standards, safety PLCs are used, which contain diagnostic functions to detect internal faults in the hardware and avoid unsafe situations that could be caused by such faults. For a safety PLC implementation, the supervisory controller has to be split into a regular and a safety part. In [27], a method is presented that automatically performs this split. It has been used to generate controller code for the Oisterwijksebaanbrug, a rotating bridge in the Wilhelmina canal at the city of Tilburg, which was then successfully employed for real-life operation of the bridge [32]. This application demonstrated all the necessary steps to go from a specification to an implementation of a supervisor, driving home the point that supervisory control theory is a viable engineering approach to transform a consistent and complete set of well-defined requirements into efficient PLC code for controlling industrial infrastructural systems.

Tooling The reported case studies have all been carried out using the CIF 3 toolset [1]. It is based on networks of hybrid automata with invariants and differential algebraic equations. Automata can interact in several ways: multi-party synchronization via shared events; and shared variables (local write, global read). CIF 3 supports a rich set of data types and expressions (e.g., lists, sets, dictionaries, and tuples), functions, conditional updates, and multi-assignments. Large-scale systems can be modeled conveniently owing to parametrized process definitions and instantiations of automata, grouping of arbitrary components in sub-scopes, and an import mechanism.

The synthesis algorithm of CIF 3 (based on the algorithm of [23]) computes various predicates, such as conditions under which controllable events may take place in the controlled system, and the initialization predicate of the controlled system. These predicates are included in the supervisor that results from synthesis. Crucial for the good performance of the toolset is that internally such predicates are represented compactly in the form of binary decision diagrams. Parts of the CIF 3 tool were inspired by Supremica [16], including the use of advanced model checking techniques such as symbolic representations.

The CIF 3 simulator enables interactive visualization-based simulation of the behavior of the controlled system. It can be employed to validate models in isolation. Additionally it may be used to validate the supervisory controller when put in the context of the uncontrolled hybrid plant.

Interoperability with other languages and tools is achieved by means of model transformations, external functions, and co-simulation via the Matlab/Simulink S-function interface. PLC code generation conforming to the IEC 61131-3 standard allows for implementation of CIF 3 supervisory controllers in actual systems.

From 2020 on, development of the CIF 3 toolset continues within the Eclipse project ESCET[2] (Supervisory Control Engineering Toolset), offering an open environment in which interested academic and industrial partners can collaborate on and profit from the further development of tool support for supervisory control theory.

4 Research challenges

We envision the following research challenges for the coming years.

- *Stronger liveness notions*: Reachability of a marked state by the plant is a rather limited notion to ensure liveness properties, which express that eventually something good will happen. It might be a good idea to integrate stronger liveness notions in the synthesis algorithms.
- *Exploit symmetry reduction*: The work on symmetry reduction in [20] may be extended from validation to optimization of synthesis algorithms. Furthermore, it could be integrated with multilevel synthesis to profit from symmetries in subclusters of components.
- *Insightful counterexample generation*: The framework in [38] can provide causes on why a certain state was trimmed by a supervisor, but often there are many such causes. Informative feedback requires a sensible selection from these causes and depicting them in an insightful manner. Moreover, counterexample generation needs to be integrated in the tooling.
- *Utilize DSMs in tools*: Structure information from a DSM regarding a plant could be exploited to optimize synthesis algorithms and tools. In particular, inside CIF 3 predicates are represented by binary decision diagrams. Their sizes are very sensitive to the ordering of systems variables against which they are constructed. In [39], it was shown that the difference between the 'right' and 'wrong' ordering for the variables in Lock III can mean a shift in computation time from a few seconds to a few hours. Initial results show that ordering variables based on a DSM can yield a significant compaction of binary decision diagrams.
- *Improve supervisory controller generation from a supervisor*: It seems possible to relax the requirements on supervisors to ensure finite response and confluence in [31]. Moreover, ideally these notions will be strengthened to ensure that a supervisory controller always eventually returns the plant to a marked state [15].
- *Integration into the engineering process*: Supervisory control synthesis needs to be made accessible for system engineers, by clear GUIs in the tools, adding

[2] https://projects.eclipse.org/projects/technology.escet

features that are needed in practice, and providing proper instruction material.

5 Conclusion

The 30th anniversary of the inception of supervisory control theory by Ramadge and Wonham was celebrated by a workshop on December 11, 2017, preceding the *CDC* conference in Melbourne, Australia [14]. The case studies presented in the current paper, in the context of a project with Rijkswaterstaat, show that this theory has reached a maturity level that allows it to be fully embraced by industry. The project initially focused on control systems for bridges and waterway locks, which may in the long run support smart waterways. It then extended to tunnels, taking into account important aspects such as emergency situations and escape routes. Recently it extended further to roadside systems, where managing and avoiding traffic jams is a serious point of concern.

A challenge lies ahead to support industrial partners in adapting synthesis-based engineering methods for their supervisory control systems. To strengthen the integration of supervisor synthesis within the engineering process, in [33], a graphical modeling method has been developed based on a library of standardized modules within movable bridges, inspired by work in [13, 17]. Subsystems of a plant are modeled by instantiating modules from the library. The method is supported by a tool. In this way supervisors have been developed for a family of 17 real-life bridges.

Key to the success of the project has been its close connection to the actual engineering process at Rijkswaterstaat and its subcontractors. We are grateful to Rijkswaterstaat for their financial support as well as their active participation in the project. In particular we thank Maria Angenent, John van Dinther, Patrick Maessen, Bert van der Vegt, and Han Vogel.

Finally, we gratefully acknowledge the groundbreaking work by the international research community of supervisory control theory, which has served as a corner stone and inspiration for the work reported here. In particular we would like to mention Knut Åkesson, Kai Cai, José Cury, Martin Fabian, Dennis Hendriks, Stéphane Lafortune, Robi Malik, Sahar Mohajerani, Thomas Moor, Max Hering de Queiroz, Peter Ramadge, Karen Rudie, Jan van Schuppen, Rong Su, and, last but not least, W. Murray Wonham.

References

1. van Beek, D., Fokkink, W., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J., Reniers, M.: CIF 3: Model-based engineering of supervisory controllers. In: 20th Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS'14. Lecture Notes in Computer Science, vol. 8413, pp. 575–580. Springer (2014)
2. Cai, K., Wonham, W.: Supervisory Control of Discrete-Event Systems. Communications and Control Engineering, Springer (2019)

3. Dietrich, P., Malik, R., Wonham, W., Brandin, B.: Implementation considerations in supervisory control. In: Synthesis and Control of Discrete Event Systems, pp. 185–201. Kluwer (2002)
4. Fabian, M., Hellgren, A.: PLC-based implementation of supervisory control for discrete event systems. In: 37th Conference on Decision and Control – CDC'98. vol. 3, pp. 3305–3310. IEEE (1998)
5. Goorden, M., Dingemans, C., Reniers, M., van de Mortel-Fronczak, J., Fokkink, W., Rooda, J.: Supervisory control of multilevel discrete-event systems with a bus structure. In: 17th European Control Conference – ECC'19. pp. 3204–3211. IEEE (2019)
6. Goorden, M., Fabian, M.: No synthesis needed, we are alright already. In: 15th Conference on Automation Science and Engineering – CASE'19. pp. 195–202. IEEE (2019)
7. Goorden, M., van de Mortel-Fronczak, J., Etman, L., Rooda, J.: DSM-based analysis for the recognition of modeling errors in supervisory controller design. In: 21st Dependency and Structure Modeling Conference, DSM'19. pp. 127–135 (2019)
8. Goorden, M., van de Mortel-Fronczak, J., Reniers, M., Fokkink, W., Rooda, J.: The impact of requirement splitting on the efficiency of supervisory control synthesis. In: 17th Conference on Formal Methods in Industrially Critical Systems – FMICS'19. Lecture Notes in Computer Science, vol. 11687, pp. 76–92. Springer (2019)
9. Goorden, M., van de Mortel-Fronczak, J., Reniers, M., Fokkink, W., Rooda, J.: Modeling guidelines for component-based supervisory control synthesis. In: 16th Conference on Formal Aspects of Component Software – FACS'19. Lecture Notes in Computer Science, vol. 12018, pp. 1–22. Springer (2019)
10. Goorden, M., van de Mortel-Fronczak, J., Reniers, M., Fokkink, W., Rooda, J.: Structuring multilevel discrete-event systems with dependency structure matrices. IEEE Transactions on Automatic Control **65**(4), 1625–1639 (2019)
11. Goorden, M., Reniers, M., van de Mortel-Fronczak, J., Fokkink, W., Rooda, J.: Compositional coordinator synthesis for discrete event systems (2020), submitted to *Discrete Event Dynamic Systems*
12. Goorden, M., van de Mortel-Fronczak, J., Reniers, M., Fabian, M., Fokkink, W., Rooda, J.: Model properties for efficient synthesis of nonblocking modular supervisors. arXiv preprint arXiv:2007.05795 (2020)
13. Grigorov, L., Butler, B., Cury, J., Rudie, K.: Conceptual design of discrete-event systems using templates. Discrete Event Dynamic Systems **21**(2), 257–303 (2011)
14. Lafortune, S., Rudie, K., Tripakis, S.: Thirty years of the Ramadge-Wonham theory of supervisory control: A retrospective and future perspectives. IEEE Control Systems Magazine **38**(4), 111–112 (2018)
15. Malik, P.: From Supervisory Control to Nonblocking Controllers for Discrete Event Systems. Ph.D. thesis, Universität Kaiserslautern (2003)
16. Malik, R., Åkesson, K., Flordal, H., Fabian, M.: Supremica–an efficient tool for large-scale discrete event systems. In: 20th IFAC World Congress. IFAC-PapersOnline **50**(1), 5794–5799 (2017)
17. Malik, R., Fabian, M., Åkesson, K.: Modelling large-scale discrete-event systems using modules, aliases, and extended finite-state automata. In: 18th IFAC World Congress. IFAC Proceedings Volumes **44**(1), 7000–7005 (2011)
18. Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional nonblocking verification of extended finite-state machines. Discrete Event Dynamic Systems **26**(1), 33–84 (2016)

19. Moor, T.: A discussion of fault-tolerant supervisory control in terms of formal languages. Annual Reviews in Control **41**, 159–169 (2016)
20. Moormann, L., Goorden, M., van de Mortel-Fronczak, J., Fokkink, W., Maessen, P., Rooda, J.: Efficient validation of supervisory controllers using symmetry reduction. In: 15th Workshop on Discrete Event Systems – WODES'20. IFAC (2020), in press
21. Moormann, L., Maessen, P., Goorden, M., van de Mortel-Fronczak, J., Rooda, J.: Design of a tunnel supervisory controller using synthesis-based engineering. In: ITA-AITES World Tunnel Congress – WTC'20 (2020), in press
22. Moormann, L., van de Mortel-Fronczak, J., Fokkink, W., Rooda, J.: Exploiting symmetry in dependency graphs for model reduction in supervisor synthesis. In: 16th Conference on Automation Science and Engineering – CASE'20. pp. 660––667. IEEE (2020)
23. Ouedraogo, L., Kumar, R., Malik, R., Åkesson, K.: Nonblocking and safe control of discrete-event systems modeled as extended finite automata. IEEE Transactions on Automation Science and Engineering **8**(3), 560–569 (2011)
24. Paoli, A., Sartini, M., Lafortune, S.: Active fault tolerant control of discrete event systems using online diagnostics. Automatica **47**(4), 639–649 (2011)
25. de Queiroz, M., Cury, J.: Modular supervisory control of large scale discrete event systems. In: Discrete Event Systems, Engineering and Computer Science, vol. 569. Springer (2000)
26. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event processes. SIAM Journal on Control and Optimization **25**(1), 206–230 (1987)
27. Reijnen, F., Erens, T., van de Mortel-Fronczak, J., Rooda, J.: Supervisory control synthesis for safety PLCs. In: 15th Workshop on Discrete Event Systems – WODES'20. IFAC (2020), in press
28. Reijnen, F., Goorden, M., van de Mortel-Fronczak, J., Reniers, M., Rooda, J.: Application of dependency structure matrices and multilevel synthesis to a production line. In: 2nd Conference on Control Technology and Applications – CCTA'18. pp. 458–464. IEEE (2018)
29. Reijnen, F., Goorden, M., van de Mortel-Fronczak, J., Rooda, J.: Supervisory control synthesis for a waterway lock. In: 1st Conference on Control Technology and Applications – CCTA'17. pp. 1562–1563. IEEE (2017)
30. Reijnen, F., Goorden, M., van de Mortel-Fronczak, J., Rooda, J.: Modeling for supervisor synthesis – a lock-bridge combination case study. Discrete Event Dynamic Systems **30**(3), 499–532 (2020)
31. Reijnen, F., Hofkamp, A., van de Mortel-Fronczak, J., Rooda, J.: Finite response and confluence of state-based supervisory controllers. In: 15th Conference on Automation Science and Engineering – CASE'19. pp. 509–516. IEEE (2019)
32. Reijnen, F., Leliveld, E.B., van de Mortel-Fronczak, J., van Dinther, J., Rooda, J., Fokkink, W.: A synthesized fault-tolerant supervisory controller for a rotating bridge (2020), under submission
33. Reijnen, F., van de Mortel-Fronczak, J., Reniers, M., Rooda, J.: Design of a supervisor platform for movable bridges. In: 16th Conference on Automation Science and Engineering – CASE'20. pp. 1298—1304. IEEE (2020)
34. Reijnen, F., Reniers, M., van de Mortel-Fronczak, J., Rooda, J.: Structured synthesis of fault-tolerant supervisory controllers. In: 10th Symposium on Fault Detection, Supervision and Safety of Technical Processes – SAFEPROCESS'18. IFAC-PapersOnLine, **51**(24), 894–901 (2018)
35. Reijnen, F., Verbakel, J., van de Mortel-Fronczak, J., Rooda, J.: Hardware-in-the-loop set-up for supervisory controllers with an application: the Prinses Marijke

complex. In: 3rd Conference on Control Technology and Applications – CCTA'19. pp. 843–850. IEEE (2019)
36. Sköldstam, M., Åkesson, K., Fabian, M.: Modeling of discrete event systems using finite automata with variables. In: 46th Conference on Decision and Control – CDC'2007. pp. 3387–3392. IEEE (2007)
37. Swartjes, L., van Beek, D., Reniers, M.: Towards the removal of synchronous behavior of events in automata. In: 12th Workshop on Discrete Event Systems – WODES'14. IFAC Proceedings Volumes **47**(2), 188–194 (2014)
38. Swartjes, L., Reniers, M., Fokkink, W.: Deducing causes for the absence of states in supervised systems. In: 6th Conference on Control, Decision and Information Technologies – CoDIT'19. pp. 144–149. IEEE (2019)
39. Thuijsman, S., Hendriks, D., Theunissen, R., Reniers, M., Schiffelers, R.: Computational effort of BDD-based supervisor synthesis of extended finite automata. In: 15th Conference on Automation Science and Engineering – CASE'19. pp. 486–493. IEEE (2019)
40. Wilschut, T., Etman, L., Rooda, J., Adan, I.: Multilevel flow-based Markov clustering for design structure matrices. Journal of Mechanical Design **139**(12), 121402 (2017)
41. Wonham, W., Ramadge, P.: Modular supervisory control of discrete-event systems. Mathematics of Control, Signals, and Systems **1**(1), 13–30 (1988)