Supervisor Synthesis: Bridging Theory and Practice

W. J. Fokkink
Vrije Universiteit Amsterdam

M. A. Goorden
Aalborg University

J. M. van de Mortel-Fronczak
Eindhoven University of Technology

F. F. H. Reijnen
Rijkswaterstaat

J. E. Rooda
Eindhoven University of Technology

***Abstract*—Supervisor synthesis is a classical approach from the eighties to automatically generate a controller for a discrete-event system, ensuring its safe operation. In recent years, owing to important improvements, the applicability of supervisor synthesis has increased significantly. We discuss some notable new developments that were pivotal in the application of supervisor synthesis to large infrastructural systems.**

■ **A PIECE OF SOFTWARE** tends to create new functional behavior. By contrast, controller software limits the behavior of a cyber-physical system by blocking possible behavior of its components. For example, in a movable bridge, the bridge deck, barriers, and traffic lights can in principle operate independently. A supervisory controller, supervisor for short, makes sure the bridge deck only opens when the barriers are closed and the traffic lights are red. Assembly lines in factories are filled to the brim with controller software. Supervisory control is also omnipresent in our daily lives, e.g., in printers, elevators, and traffic.

The main purpose of a supervisor is *compliance*: It guarantees that system behavior satisfies all safety requirements formulated by the system engineer. Safety requirements express which bad situations need to be avoided during operation of the physical system, called the plant. A supervisor should also be *minimally restrictive*: It blocks the minimal amount of behavior to achieve compliance. A human operator can then further guide the plant in its operation. For example, a bridge is typically operated by a human, ever more often remotely, to stop traffic and open the bridge deck when ships need to pass through. The supervisor safeguards this operation, preventing various human errors, for instance by blocking an event (i.e., action) that would open the bridge deck while the barriers are not yet closed.

Figure 1 depicts the general layout of such a control structure. At the top a human operator regulates the plant through a graphical user interface, at the bottom are the mechanical plant components. Sensors inform the operator of the plant's status, whereas actuators drive the operation of plant components. Resource control, not shown in the picture, is needed to regulate electromechanical actuators based on information

Published by the IEEE Computer Society

from associated sensors. The supervisor in the middle makes sure actuator signals that would violate safety requirements are disabled.

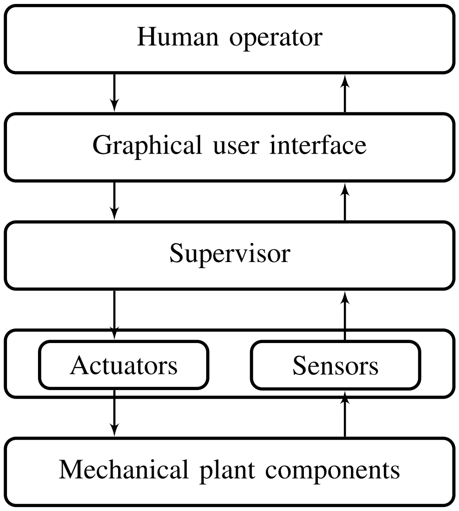

Figure 1: Schematic view of a control structure.

The SCADA (supervisory control and data acquisition) framework, widely used in industry, ranges from high-level process management such as graphical user interfaces and networked data communication all the way down to controllers that interface with the plant and programmable logic controllers at the hardware level.

The state space of a plant, representing its potential behavior, tends to be huge, growing exponentially with the number of plant components. The controller needs to work correctly on the entire state space, also in corner cases that rarely occur. A human developer of controller software needs to have deep insight into the plant's behavior as well as its safety requirements to decide which events of the different plant components need to be blocked and in which situations. Therefore, there is a need for automatically and efficiently constructing controller software.

SUPERVISOR SYNTHESIS

In 1987, Ramadge and Wonham [1] put forward an ingenious idea to automatically generate controller software for a discrete-event system, given precise descriptions of the behavior of the plant components as well as the safety requirements for the overall plant behavior.

Both plant components and safety requirements can be specified conveniently in the classical formalism of finite automata, consisting of a finite number of states and transitions between these states. Each transition carries an event: A transition from state s to state s' with event e means that from state s one can move to state s' by performing event e. Decorating the states and events in a finite automaton with variables and allowing Boolean guards on transitions, specifying under which conditions the transitions can take place, facilitates the specification of real-life systems. This is called an extended finite automaton, abbeviated to EFA.

Basically, the plant components together with the requirements perform a perfect choreography, called their synchronous product. Each event leads to an update in the local states of the plant components and requirements that are sensitive to this event. If an event is missing in the local states of one or more of these plant components, then it is physically absent from the overall system state. If on the other hand an event is missing only in the local states of one or more of these requirements, then it is physically possible but must be blocked by the supervisor.

Not all events are under the control of the supervisor. Typically, output events such as signals to actuators can be prevented but input events such as signals from sensors cannot. The first category of events is called controllable, the second category uncontrollable. A synthesized supervisor must be *controllable*: It blocks only controllable events. If an uncontrollable event must be prevented by the supervisor, the system state where it occurs is made unreachable by the supervisor, by blocking all controllable events leading to it. Moreover, if an uncontrollable event leads to this state, the origin state of this event must be made unreachable too, since the event cannot be blocked by the supervisor.

The top part of **Figure 2** depicts EFAs of two plant components that are part of a waterway lock: (a) an actuator of a motor for a lock gate and (b) an equal water level sensor. Circles are locations; a state consists of a location together with a valuation of variables. Each EFA has one initial state, indicated by a short horizontal incoming arrow. In state On the lock gate can be moved, while in state Off this is not possible. In state Yes the sensor detects an equal water level, while in state No it does not. Controllable events c_on and c_off

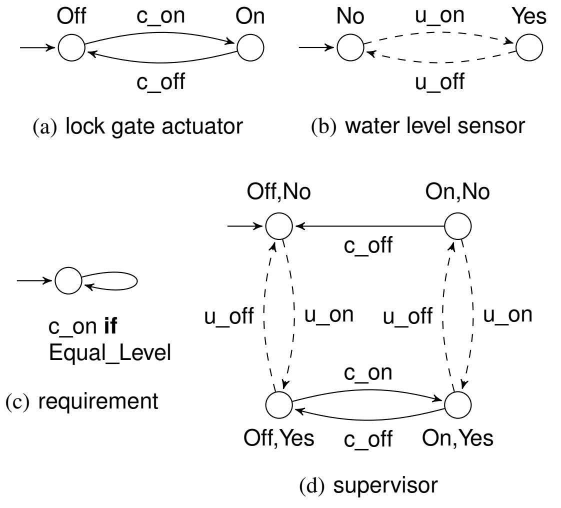

Figure 2: Part of a waterway lock.

of the lock gate actuator are represented by solid arrows and uncontrollable events u_on and u_off at the sensor by dashed arrows. Requirement (c) expresses that the actuator can only move the lock gate when the sensor measures an equal water level. The if condition in this requirement refers to a Boolean variable Equal_Level that is true only in state Yes of the sensor component. We note that the EFA for the actuator in (a) is not sensitive to (i.e., does not contain events) u_on and u_off, so that it does not block these events in the synchronous product. Likewise, the sensor in (b) is not sensitive to c_on and c_off, and the requirement in (c) is only sensitive to c_on. Finally, (d) depicts the automatically generated supervisor. Note that event c_on from the left top to the right top state, present in the synchronous product of the two plant components, is disabled by the supervisor because it is not allowed by the requirement.

Next to safety requirements, it is important to guarantee liveness properties for the plant, expressing that something good can always eventually happen. Consider once again the example of a bridge. If the bridge deck remains open forever, with the barriers closed and the traffic lights displaying red, all safety requirements are nicely satisfied. But for correct operation of the bridge, the human operator must be able to close the bridge, so that traffic over it can be resumed. Such liveness properties are captured by the notion of marked states in EFAs. A supervisor must be *nonblocking*: The plant can always return to a system state consisting of marked local states of the plant components and of the safety requirements. In the bridge example, marked local states of plant components would typically be “closed” for the bridge deck, “open” for the barriers, and “green” for the traffic lights. The controller has the difficult task of recognizing that from a reachable system state s one cannot reach a marked system state anymore. If so, it must make s unreachable.

Automatically generating a compliant, minimally restrictive, controllable, and nonblocking supervisor from specifications of the plant components and the safety requirements is referred to as *supervisor synthesis* [2]. Its main bottleneck is the dreaded state space explosion problem. Supervisor synthesis in principle needs to chart out the entire state space to determine which controllable events need to be blocked when. But this is unfeasible for real-life plants, as their number of states tends to dwarf the number of atoms in the universe.

We discuss recently developed methods that help supervisor synthesis cope with such large state spaces. We also touch upon fault-tolerant supervisors and tool support. Applications of supervisor synthesis to large infrastructural systems, notably waterway locks and bridges, highlight the strength of the approach.

MULTILEVEL SYNTHESIS

In multilevel synthesis [3], plant components and requirements are grouped together in a node of a tree. The tree structure tends to resemble closely a system decomposition, often favored by engineers to cope with the sheer size of cyber-physical systems. Requirements with many events in common are clustered in the same node, and each plant component that is sensitive to one or more events present in these requirements is added to this node. A localized supervisor is generated for each node separately. If a requirement is imposed on one cluster in the tree, then the synchronous product of all nodes assures it is imposed on the entire operation of the plant. This means the overall supervisor can be composed of the localized supervisors. By making sure each cluster concerns only few plant components, the localized supervisors remain relatively small, compared to one monolithic supervisor.

In [4] it was shown how an efficient tree structure can be obtained through a design structure matrix (DSM) [5] that registers the number of shared requirements between each pair of plant components. An algorithm from [6] reorders the plant components in such a way that shared requirements tend to move close to the diagonal of the DSM, so that tightly coupled plant components are placed side by side. Based on this DSM, different requirements that tend to relate to the same plant components are grouped together. Requirements and plant components can be clustered effectively based on this grouping. Experimental results on real-life systems presented in [4] show substantial reductions in the size and generation time of multilevel supervisors, compared to their monolithic counterparts.

Although the localized supervisors are nonblocking, the overall supervisor may not be. It needs to be guaranteed that all localized supervisors can always reach a marked state at the same moment in time. This can be verified by performing a sequence of transformations on the localized supervisors to obtain a simplified monolithic EFA, which is blocking if and only if the overall supervisor induced by the initial localized supervisors is blocking [7]. In case it is blocking, the simplified monolithic EFA contains precisely enough information on how to prevent blocking. Reversals of those transformation steps can be employed to obtain a coordinator that ensures the localized supervisors can always reach a marked state simultaneously [8].

Multilevel synthesis based on a DSM has two vulnerabilities. First, real-life plants often contain so-called bus components with many connections across the system, leading to a top-heavy tree. This can be resolved by excluding bus components from the DSM and adding one extra branch to the tree, dedicated to requirements that relate only to bus components [9]. Second, system engineers tend to blend related requirements or plant components into one. Splitting them leads to much more efficient multilevel synthesis [10].

A case study on a waterway lock in the Wilhelmina canal at the city of Tilburg, raising and lowering vessels between two water levels, shows the strength of multilevel synthesis and its optimizations. The monolithic supervisor has $6.0 \cdot 10^{24}$ states, its multilevel counterpart $1.8 \cdot 10^{23}$ states. Splitting the 142 requirements into 358 requirements and the 35 plant components into 51 smaller components leads to a significantly reduced multilevel supervisor of $2.9 \cdot 10^{10}$ states. Finally, treating stop buttons as special bus components reduces it further to only $7.7 \cdot 10^{8}$ states.

FAULT-TOLERANT SUPERVISORS

A supervisor should continue to operate, with degraded service, when a fault occurs, such as a broken wire, defect sensor, or blocking actuator. One option is to activate another supervisor, designed specifically for the type of fault that occurred [11]. Alternatively, the plant components and requirements can be divided into fault-free and post-fault behavior, depending on the type of fault. The latter approach was used in [12] to synthesize a fault-tolerant supervisor for the Algera bascule bridge over the river IJssel, depicted in **Figure 3**, with two vehicle roadways as well as lanes for cyclists and pedestrians.

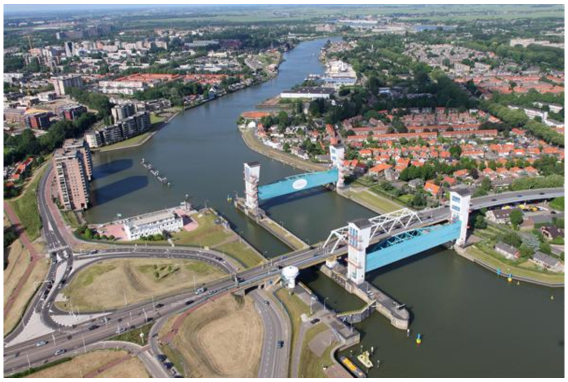

Figure 3: The Algera complex consisting of a lock, a bridge, and two storm surge barriers [https://beeldbank.rws.nl, Rijkswaterstaat / Joop van Houdt].

Supervisory controllers are often implemented using programmable logic controllers (PLCs). In industry, supervisors must adhere to strict safety standards. Therefore, safety PLCs are widely used, containing diagnostic functions to detect internal faults in the hardware and avoid unsafe situations that could be caused by such faults. A safety PLC implementation can be generated automatically from a (synthesized) supervisor [13]. This method was used to generate controller code for the Oisterwijksebaanbrug, a rotating bridge in the Wilhelmina canal, which was successfully employed for real-life operation of the bridge [14]. This application indicates that supervisor synthesis is a viable engineering approach to transform a requirements specification into efficient PLC code for controlling a real-life system.

After a system failure, reconstructing and analyzing the behavior that led to this undesired situation provides insight into the cause. Incident analysis for PLCs usually consists of plotting actuator and sensor signals, which is laborious and yields data that may be difficult to interpret. In [15], a method was proposed in which an EFA that captures the actual system behavior is constructed via simulation. By comparing this EFA with the EFAs that specify the supervised plant behavior, faults can be identified more easily. This approach was successfully applied to the aforementioned rotating bridge.

TOOLS

Two well-established tools for supervisor synthesis are Supremica [16] and CIF 3 [17]. They both use the synthesis algorithm from [18], based on iterative strengthening of guards on transitions so that forbidden states become unreachable in the controlled plant. Guards are represented and manipulated symbolically in the form of binary decision diagrams [19].

The reported case studies were carried out using CIF 3. Large-scale systems can be modeled conveniently owing to parametrized plant component definitions. A simulator enables interactive visualization-based simulation of the behavior of the controlled plant. Efficient PLC code can be generated automatically [20]; this code conforms to the IEC 61131-3 standard.

From 2020 on, development of the CIF 3 toolset continues within Eclipse Supervisory Control Engineering Toolset (ESCET™), offering an open environment in which interested academic and industrial partners can collaborate on and profit from the further development of tool support for supervisor synthesis.[1]

CONCLUSION

Supervisory control synthesis has risen to a level where it can be successfully applied in the engineering process of real-life discrete-event systems. Multi-level synthesis based on DSMs makes it possible to automatically synthesize supervisors and generate control software for large infrastructural systems. Important optimizations to increase its effectiveness are treating bus components separately and splitting requirements as well as plant components. Fault-tolerance can be taken into account both at supervisor and at PLC level.

Next to bridges and waterway locks, supervisor synthesis is within Rijkswaterstaat also being applied in the development of controller software for road tunnels. The potential application domain is not limited to infrastructural systems but covers discrete-event systems at large. For example, supervisor synthesis was successfully applied to advanced driver assistance systems, theme park vehicles, and MRI scanners.

Future directions for research are the development of further optimizations for supervisor synthesis as well as validation techniques for generated controller software, and supporting industry with integrating the supervisor synthesis method into their discrete-event system engineering methods.

ACKNOWLEDGMENTS

Rijkswaterstaat, part of the Dutch Ministry of Infrastructure and Water Management, provided financial and technical support over the years. Special thanks go to Han Vogel and Maria Angenent. We gratefully acknowledge Pascal Etman, Martin Fabian, and Michel Reniers for their research contributions.

■ REFERENCES

1. P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
2. W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, Springer, 2018.
3. J. Komenda, T. Masopust, and J. H. van Schuppen, "Control of an engineering-structured multilevel discrete-event system," *Proc. 13th Workshop Discrete Event Systems*, pp. 103–108, IEEE, 2016.
4. M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, and J. E. Rooda, "Structuring multilevel discrete-event systems with dependency

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

structure matrices," *IEEE T. Automat. Contr.*, vol. 65, no. 4, pp. 1625–1639, 2020.

5. S. D. Eppinger and T. R. Browning, *Design Structure Matrix Methods and Applications*, MIT Press, 2012.
6. T. Wilschut, L. F. P. Etman, J. E. Rooda, and I. J. B. F. Adan, "Multilevel flow-based Markov clustering for design structure matrices," *J. Mech. Des.*, vol. 139, no. 2, 121402, 2017.
7. S. Mohajerani S, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *Discrete Event Dyn. Syst.*, vol. 26, no. 1, pp. 33-–84, 2016.
8. M. A. Goorden, M. Fabian, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, and J. E. Rooda, " Compositional coordinator synthesis of extended finite automata," *Discrete Event Dyn. Syst.*, vol. 31, no. 3, 2021.
9. 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," *Proc. 18th European Control Conf.*, pp. 3204–3211, IEEE, 2019.
10. M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, and J. E. Rooda, "The impact of requirement splitting on the efficiency of supervisory control synthesis," *Proc. 24th Conf. Formal Methods for Industrial Critical System*, pp. 76–92, Springer, 2019.
11. A. Paoli, M. Sartini, and S. Lafortune, "Active fault tolerant control of discrete event systems using online diagnostics," *Automatica*, vol. 47, no. 4, pp. 639–649, 2011.
12. F. F. H. Reijnen, M. A. Reniers, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Structured synthesis of fault-tolerant supervisory controllers," *Proc. 10th Symp. Fault Detection, Supervision and Safety of Technical Processes*, pp. 894–901, IFAC, 2018.
13. F. F. H. Reijnen, T. Erens, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory control synthesis for safety PLCs," *Proc. 17th Workshop Discrete Event Systems*, pp. 151–158, IFAC, 2020.
14. F. F. H. Reijnen, E.-B. Leliveld, J. M. van de Mortel-Fronczak, J. van Dinther, J. E. Rooda, and W. J. Fokkink, "Synthesized fault-tolerant supervisory controllers, with an application to a rotating bridge," *Comput. Ind.*, vol. 130, 103473, 2021.
15. F. F. H. Reijnen, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Data logging and reconstruction of discrete-event system behavior," *Proc. 16th Conf. Control, Automation, Robotics and Vision*, pp. 1020–1026, IEEE, 2020.
16. R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica–An efficient tool for large-scale discrete event systems," *Proc. 20th IFAC World Congr.*, pp. 5794–5799, IFAC, 2017.
17. D. A. van Beek et al., "CIF 3: Model-based engineering of supervisory controllers," *Proc. 20th Conf. Tools and Algorithms for the Construction and Analysis of Systems*, pp. 575–580, Springer, 2014.
18. L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE T. Autom. Sci. Eng.*, vol. 8, no. 3, pp. 560–569, 2011.
19. K. L. McMillan, *Symbolic Model Checking*, Springer, 1993.
20. F. F. H. Reijnen, T. R. Erens, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory controller synthesis and implementation for safety PLCs," *Discrete Event Dyn. Syst.*, 2022.

Wan Fokkink received a Ph.D. degree from the University of Amsterdam. He is professor of Theoretical Computer Science at the Vrije Universiteit and professor of Model-Based System Engineering at Eindhoven University of Technology. His research concerns the analysis of distributed computer systems. Contact him at w.j.fokkink@vu.nl.

Martijn Goorden received a Ph.D. degree from Eindhoven University of Technology. He is postdoctoral researcher at Aalborg University. His research concerns model-based engineering. Contact him at mgoorden@cs.aau.dk.

Joanna van de Mortel-Fronczak received a Ph.D. degree from Eindhoven University of Technology, where she currently is assistent professor at the Faculty of Mechanical Engineering. Her research concerns model-based engineering. Contact her at j.m.v.d.mortel@tue.nl.

Ferdie Reijnen received a Ph.D. degree from Eindhoven University of Technology. He is industrial automation consultant at Rijkswaterstaat. His research concerns model-based system design. Contact him at ferdie.reijnen@rws.nl.

Jacobus Rooda received a Ph.D. degree from the University of Twente. He was Professor of (Manufacturing) Systems Engineering at Eindhoven University of Technology since 1985. As Professor Emeritus since 2010 he still actively researches engineering design for industrial systems. Contact him at

j.e.rooda@tue.nl.