2018 IEEE Conference on Control Technology and Applications (CCTA)
Copenhagen, Denmark, August 21-24, 2018

Application of Dependency Structure Matrices and Multilevel Synthesis to a Production Line

F.F.H. Reijnen[1], M.A. Goorden[1], J.M. van de Mortel-Fronczak[1], M.A. Reniers[1] and J.E. Rooda[1]

Abstract— **Designing correct supervisory controllers for high-tech systems is becoming increasingly complex due to demands for verified safety, higher quality, and more functionality. Based on supervisory control theory, a method is defined to automatically derive a supervisor from a model of the uncontrolled system (the plant) and a model of the control requirements. A drawback of this approach is the computational complexity that grows exponentially with the number of components in the system. Several control architectures have been proposed in the literature to overcome this problem. It is not always evident which control architecture is most suitable for a given system. Therefore, the research on methods supporting automatic architecture generation gains in importance. The purpose of this paper is to illustrate such a method in a case study involving the development of supervisory control. The control architecture is automatically derived from the plant and requirement models, using clustering techniques for dependency structure matrices. The following development steps are discussed: modeling the plant, modeling the requirements, structuring the models in a multilevel control architecture, synthesizing the supervisors, and generating a real-time controller for implementation.**

I. INTRODUCTION

Complexity growth of high-tech systems is caused by increasing market demands for verified safety, higher quality, and extensive functionality. At the same time, it is desired to decrease costs and time-to-market. Model-based systems engineering methods, such as supervisor synthesis [1], can help in coping with this complexity when designing supervisory controllers. Models allow for early verification and validation, such that fewer design errors are found in the later implementation and testing phases [2].

Supervisor synthesis [1] is a method to automatically derive a supervisor from a model of the plant (modeled as a collection of components) and a model of the control requirements. The synthesized supervisor controls the plant such that the controlled system adheres to the requirements per construction. A drawback of supervisory control theory is the computational complexity that grows linearly with the number of states represented by the plant model, which grows exponentially with the number of components in the system, referred to as state space explosion. Several methods exist that try to overcome this complexity by dividing the state space into modules [3]–[6].

One such method is multilevel discrete-event control synthesis [6], this method can help in reducing the usage of computational resources in the supervisor development process. For multilevel synthesis, the system is divided in a tree-based structure, where each subsystem has a set of children and a unique parent. Each subsystem consists of a sub-collection of the components models and a sub-collection of the requirement models. A supervisor is synthesized for each subsystem in the tree using monolithic synthesis.

In [7], a method is shown which automatically transforms any collection of component models and collection of requirement models into a multilevel discrete-event system. This transformation uses analysis techniques for dependency structure matrices (DSMs) [8] to derive the system structure from the component models and requirement models. Implementation code can automatically be generated from the collection of synthesized supervisors, using the approach from [9]. This way of working is schematically depicted in Fig. 1.

Even though supervisory control theory has been widely accepted in academia, and numerous applications have been proposed in the literature, realistic applications are yet few in number [10]. Two successful attempts at synthesizing and implementing a supervisor are reported for a theme park vehicle in [11], and for a patient support system for an MRI scanner in [12]. In [6] and [7], both cases are analyzed to show the feasibility of multilevel synthesis. However, implementation on hardware was not considered there. To the best of our knowledge, a multilevel supervisor has never been implemented on hardware.

This paper deals with the design and implementation of a multilevel supervisory controller for a small-scale production line. The production line is a didactic system with industrial components and industrial-size complexity. It consists of five times as many components and has to adhere to more requirements than the two aforementioned systems. Didactic systems such as this one provide researchers and students with a realistic platform to analyze techniques from academia in practice. Similar systems, such as xCPS [13] and MOPED [14], are successfully used to experiment with supervisory control, image-based control, embedded software, scheduling, and real-time control.

The contribution of this paper is to show how multilevel synthesis together with DSM-based clustering techniques can be used to design supervisory controllers for a production line. The whole development process, i.e., modeling the plant and requirements, automatically structuring the models in a multilevel control architecture, synthesizing the supervisors, and implementing the supervisors on the hardware, is shown. Furthermore, the multilevel supervisor architecture

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

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

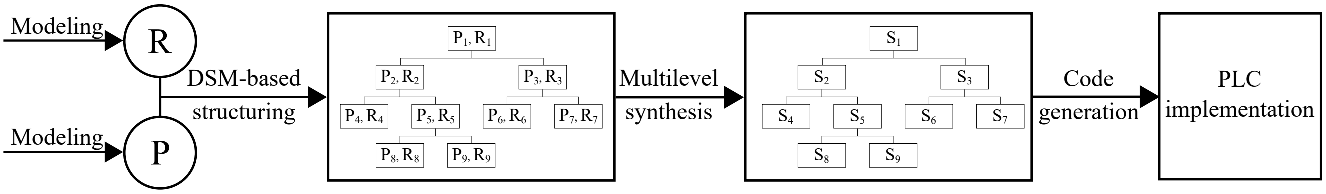

Fig. 1. Schematic way of working for DSM-based structuring, multilevel synthesis, and code generation. P denotes the collection of component models, P_i a sub-collection of the component models, R the collection of requirement models, R_i a sub-collection of the requirement models, and S_i a supervisor.

is compared to the monolithic supervisor architecture, and to a multilevel supervisor architecture obtained from manual structuring.

This paper is structured as follows. The production line analyzed in this case study is introduced in Section II. The concepts and notions of supervisor synthesis, multilevel synthesis, and DSMs are provided in Section III. Section IV discusses our modeling of the production line. In Section V, the results of DSM-based structuring are discussed. The resulting multilevel supervisor is shown and compared to supervisors obtained by two other methods in Section VI. The generation of the real-time controller and the implementation on the production line are presented in Section VII. Finally, Section VIII concludes the paper.

II. Case study description

For this case study, a small-scale production line consisting of six stations has been considered, see Fig. 2. The hardware of the system is produced by Festo Didactic (`www.festo.com`).

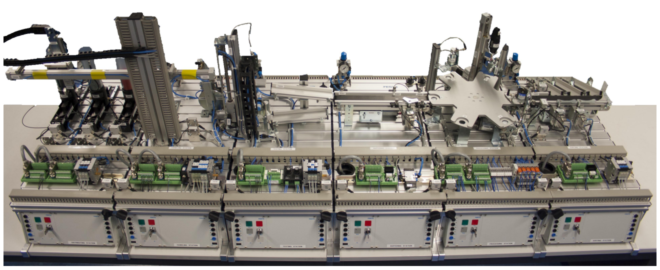

Fig. 2. Overview of the small-scale production line.

The production line is developed for vocational training in the field of automation and communication for both bachelor and master students in Mechanical Engineering. Although the system is made of industrial components, no real processing takes place. Still, all the movements, velocities, and timings are as if there was. Various types of pneumatic cylinders, and DC motors are used as actuators. Sensors of types optical, inductive, capacitive, electromechanical, reed contact, and Hall-effect are used. In total, the production line contains 28 actuators to be controlled by a supervisor and 59 sensors.

The desired controlled behavior of the production line is as follows. Products enter the production line via the *distributing* station, where they have initially been placed in three storage tubes. Products leave the storage tubes via pushers. A pneumatic gripper, belonging to the *handling* station, transports the products to the intermediate buffer. From the buffer, a transfer cylinder picks up the products and places them on the elevator plateau at the testing station. At the *testing* station, the height of the products is measured, and correct products are moved via a pneumatic slide towards the buffering station. Rejected products are placed in a local buffer. The *buffering* station consists of a conveyor belt and a separator. The separator can interrupt the supply of products towards the processing station. The *processing* station consists of a turntable with six places: entry, testing, drilling, exit, and two spare locations. Product orientation is checked at the testing location, after which the products are (virtually) machined by a drill and moved to the sorting station. The *sorting* station is the final station in the production line. Its purpose is to store the products in one of the three buffers, depending on color and material. Here, a conveyor belt transports the products. Two pneumatic gates can be opened or closed to divert products from the belt into a buffer.

The supervisor of the production line is to be implemented on a PLC, which is the industrial standard for implementing supervisory controllers. Here, a soft PLC, connected via Ethernet to six remote I/Os, is used to run the PLC code. This set-up is schematized in Fig. 3.

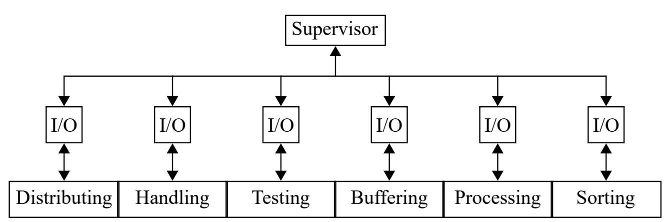

Fig. 3. Schematic overview of the stations in the production line.

III. Preliminaries

A. Supervisor synthesis

Supervisor synthesis [1] is a method to automatically derive a supervisor for a system from a model of the plant, and a model of the control requirements. The supervisor coordinates the plant such that the controlled system is guaranteed to satisfy the following properties:

- Safety. The controlled system cannot reach states or enable events that are forbidden by the requirements.

- Controllability. Only controllable events are restricted by the supervisor.
- Non-blockingness. The controlled system is always able to reach a marked state.
- Maximal permissiveness. The supervisor imposes the minimal restriction to satisfy safety, controllability, and non-blockingness.

Finite state automata (FSAs) are generally used to model the behavior of the plant. An FSA consists of a finite set of locations Q with transitions between them. These transitions are labeled by events $\sigma \in \Sigma$. The transition relation is given as $\delta \subseteq Q \times \Sigma \times Q$. Some locations are marked to indicate that they are safe; they are denoted by $Q_m \subseteq Q$. Initially, the system is in location $q_0 \in Q$, the initial location. The event set Σ is partitioned into controllable events Σ_c and uncontrollable events Σ_{uc}. Controllable events can be restricted by the supervisor, e.g., starting an actuator, whereas uncontrollable events cannot be restricted, e.g., that a sensor switches on.

Usually, the plant is modeled as a collection of several interacting components P_i, $i \in I$, with I a finite index set. The plant P is represented by the synchronous product $P = \|_{i \in I} P_i$ [15].

FSAs can be used to model the control requirements as well; this is especially useful to model the required order of events. Often, state-based expression models [16] are also used. These expressions state under which conditions a certain event is allowed to occur. For an FSA named A with location q, $A.q$ is used in expressions to denote that the current location of automaton A is q. By being able to refer to states in these expressions, the requirements can be formulated very concisely. Usually, requirement model R consists of a collection of FSAs and state-based expressions, $R_j, j \in J$, with J a finite index set.

The algorithm described in [17] can be used to synthesize a monolithic supervisor S that adheres to the aforementioned properties from P and R. To compactly represent the supervisor, the method of [18] can be used, which displays the supervisor as a finite set of guards for the controllable events. In CIF 3 [19], this resulting set of guards is returned as an extended finite state automaton, such that it is possible to compute the synchronous product. The behavior of the supervised system is then the synchronous product of the collection of components P, collection of requirements R, and supervisor S.

B. Multilevel discrete-event control synthesis

For the design and engineering of large-scale systems, the system is often decomposed into multiple subsystems. This approach is also applicable for discrete-event systems. A *multilevel discrete-event system* (MLDES) is a discrete-event system with a tree-based structure, as first proposed in [6]. Each node in the tree has a unique parent (except the top node) and may have children. Each node in the MLDES is interpreted as a subsystem. Therefore, each node consists of a sub-collection of the component models and a sub-collection of the requirement models.

To control an MLDES with node set N, for each node $n \in N$ a supervisor S_n is synthesized such that the synchronous product of all controlled subsystems equals the system controlled by a monolithic supervisor. Supervisor S_n can be obtained by synthesizing with the aforementioned algorithm a monolithic supervisor for the subsystem at node n. It can be shown that this set of supervisors satisfies the safety and controllability property [6]. When the component and the requirement models are prefix closed, the obtained result is also non-blocking and maximally permissive [15].

First applications of synthesis for MLDES show a reduction in the computational complexity and a reduction in the state space sizes of the controlled systems, see [6], [7].

C. Dependency structure matrices

This subsection provides the general concepts and notions of dependency structure matrices. A more in-depth introduction is given in [8]. A DSM is a visual representation of the linkage between elements in a system in the form of a square matrix. The same system elements are labeled along both axes of the matrix. Filled off-diagonal elements are used to indicate relationships between elements.

DSMs can be clustered, by reordering the matrix elements, to reveal modules of tightly related components. The algorithm of [20] allows for automatic clustering. DSMs have also been proven useful in revealing the multilevel structure of a plant model [7]. The method described in [7] allows for the automatic transformation of the general supervisory control problem into the multilevel control problem of [6].

IV. Modeling

In this section, the component models and the requirement models of the components belonging to the *processing* station are described in detail. Due to the large number of components and requirements in the production line, the models of the other stations are not included here, but are available in [21]. The processing station is depicted in Fig. 4.

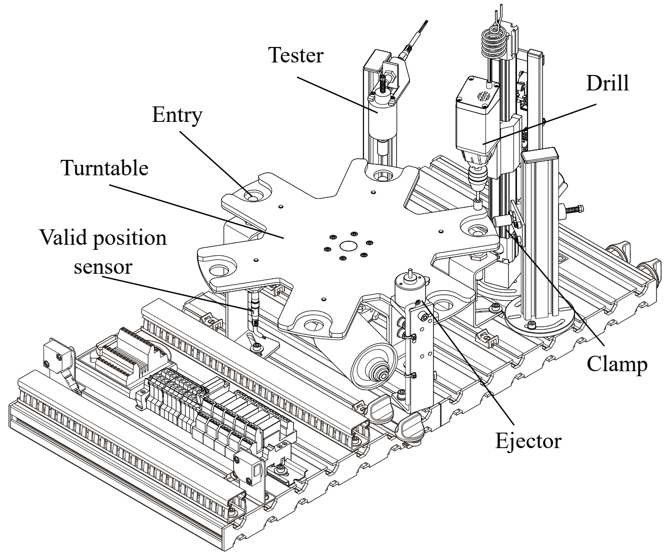

Fig. 4. The processing station [source: Festo].

The processing station consists of a turntable on which products are loaded. From the entry, they first visit the tester, which tests the orientation of the product. Subsequently, they visit the drill unit, where a product is first clamped before the drill starts machining the product. At the exit, an ejector removes the products. The turntable has two spare positions that are not used.

A. Modeling the components of the processing station

The modeling of the components is based on the inputs and outputs of the control unit. This level of detail is chosen, such that the synthesized supervisors can be directly implemented in the control unit. This results in component models for all sensors and actuators of the stations.

Four sensors measure the presence of a product at the entry, test, drill, and exit position. There is a sensor that measures if the turntable is at a valid stopping position. Two sensors measure the vertical end positions of the drill. Another sensor indicates whether a product is clamped. The tester is equipped with a sensor that checks whether the product dimensions are correct. Lastly, an internal sensor indicates whether the station is initialized. All of these sensors, except for the drill position sensors, are modeled as a simple sensor that is either *on* or *off* (see the top right automaton in Fig. 5). The unconnected incoming arrow denotes the initial location. The filled circle indicates the marked location. Solid and dashed arrows represent controllable and uncontrollable events, respectively. The drill position sensors are modeled as the bottom automaton in Fig. 5. This model includes the physical restriction that both sensors can never be on simultaneously.

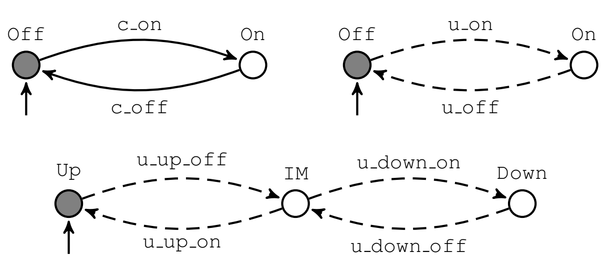

Fig. 5. Model of an actuator (top left), a sensor (top right), and the drill position sensor (bottom).

Seven actuators are controlled by the supervisor. The turntable can be rotated clockwise, the tester can be activated, the drill can be activated, the clamp can be enabled, and the ejector can be activated. For the turntable, the hardware prevents it from stopping at an invalid position; hence, the actuator only sends a short pulse to start the movement. Two actuators are used for the movement of the drill, one for ascending and one for descending. All these actuators can be modeled in the same way as the top left automaton in Fig. 5.

B. Modeling the requirements of the processing station

In order to operate the processing station in a safe and efficient manner, the following informal requirements have to be satisfied:

1) The actuators can only activate after the station has been initialized.
2) The turntable can only rotate when the other actuators are in a safe position, i.e., the drill in the upper position, the clamp retracted, and the tester, drill, and ejector deactivated.
3) The turntable can only rotate when a new product has entered.
4) The turntable actuator can only be disabled when the turntable leaves the valid position.
5) The tester, drill, clamp, and ejector are only allowed to enable when the turntable is at stand-still and there is a product.
6) The clamp can only be released when the drill is up and disabled.
7) The drill is only allowed to activate and descend when there is a clamped product.
8) The ejector can only be retracted after the product has been removed.
9) The tester can only be disabled once the measurement has been completed.
10) If there is a product at the testing or drilling location, it should be processed before the turntable is activated again.
11) The cycle at the drilling location should be as follows: clamp product, activate drill, descend, ascend, deactivate drill, release product.

Requirements 1 until 9 state conditions under which an event is allowed to occur. These requirements can be modeled intuitively using state-based models, as given below for the first four requirements. In the models, e `needs` G denotes that an event e can only occur when G evaluates to *true*. The other requirement models are similar and are available in [21].

```
R1: {A_turntable.c_on, A_drill.c_on,
     A_tester.c_on, A_drilldown.c_on,
     A_eject.c_on} needs
       S_DInitialized.On

R2: {A_turntable.c_on} needs
       S_drill.Up and S_clamp.Off and
       A_tester.Off and A_drill.Off and
       A_eject.Off

R3: {A_turntable.c_on} needs S_atentry.On

R4: {A_turntable.c_off} needs S_turntable.Off
```

Requirements 10 and 11 describe the flow in the station, i.e., they describe the order of events, and can be modeled using FSAs. For the testing location, Requirement 10 is given in Fig. 6.

Requirement 10 states that after the tester has been used, the turntable must first rotate before a new test is allowed. If there is no product, the turntable can rotate again, denoted by the self-loop in location `A`. There is a similar model for the order of events at the drill location. Models of Requirements 10, and 11 are available in [21].

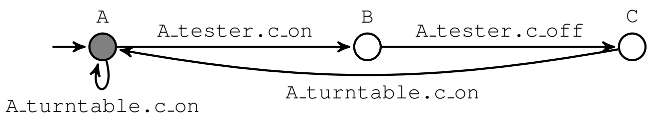

Fig. 6. Model of requirement 10 for the testing location.

V. DSM-BASED STRUCTURING

In Fig. 7, the result of clustering the component DSM, using the automated method of [7], is shown. On the axes, all 77 component models are labeled. The off-diagonal marks indicate a relation between components, via requirements. The component models of the processing station are mainly found in cluster C_3, visualized by the red coloring. For clustering, the flow-based Markov clustering algorithm of [20] is used. The following settings are used: the expansion constant, an integer value, $\alpha = 2$, the inflation constant $\beta = 2.2$, and the evaporation constant $\mu = 2.0$. Typically, when larger clusters are desired smaller values for β and μ are chosen. It is advised in [20] to use $\alpha = 2$; only in case of a dense DSM, the value of α may be decreased to 1.

Modules of components are visualized by square boxes. In total 17 supervisors at the fourth level are found (the smallest boxes); these are typically sensors and actuators that work closely together. 4 modules are found on the third level, denoted by C_1, C_2, C_3, and C_4 in the figure; these coordinate processes in stations. Some stations are clustered together, because of their close dependencies. Cluster C_1 contains the distributing station and the gripper of the handling station, cluster C_2 the transfer of the handling station, the testing station, and the buffering station. Clusters C_3 and C_4 contain the processing and sorting station, respectively. Then, there are 2 modules and 1 module for the second and first level, respectively. At these levels, inter-station interactions are coordinated.

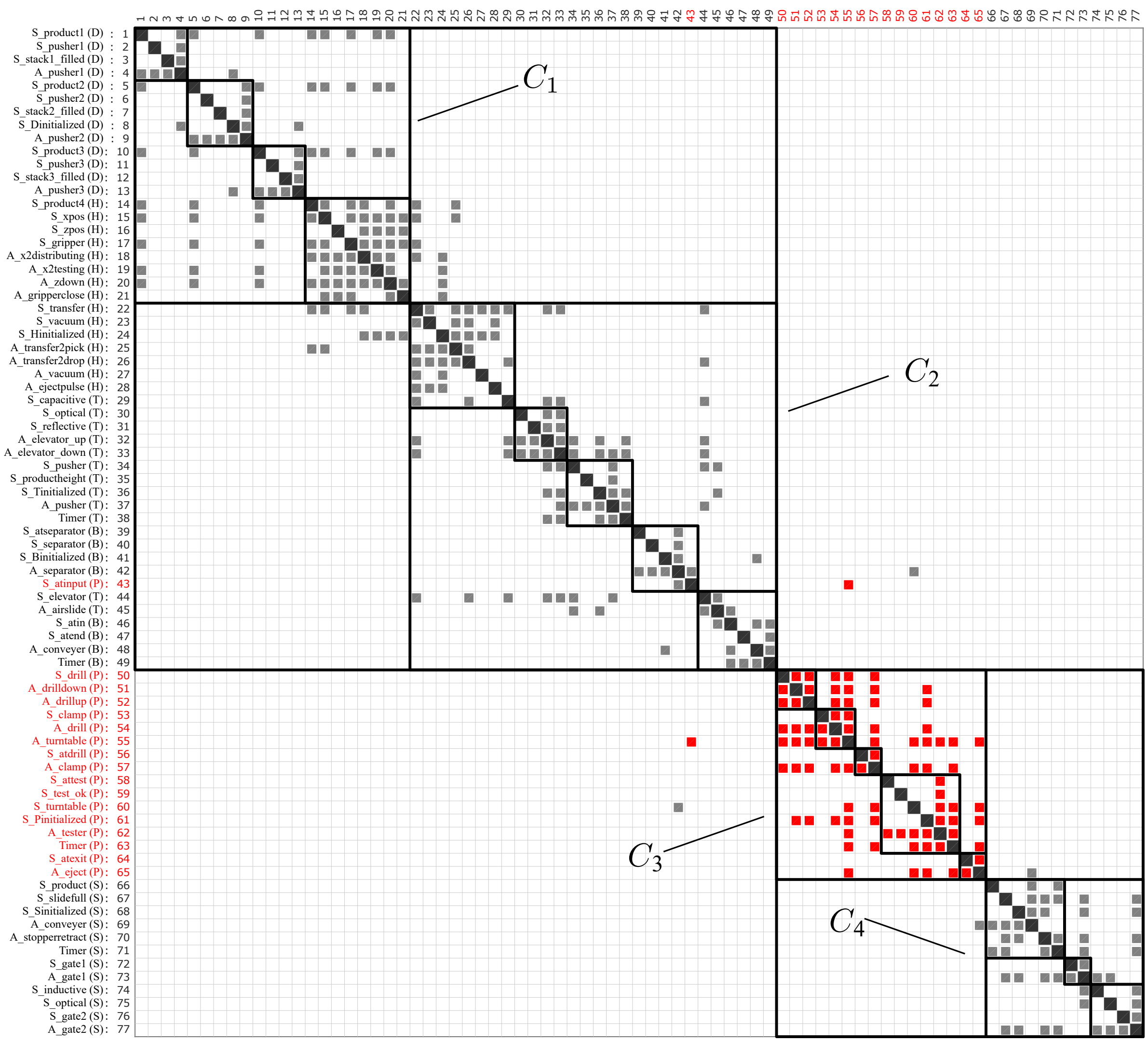

Fig. 7. The result of DSM-based clustering of the models. The relations in red are derived from the models of the processing station. C_1, C_2, C_3, and C_4 are all clusters found on the third level. Naming abbreviations: Sensor (S_), Actuator (A_), Distributing (D), Handling (H), Testing (T), Buffering (B), Processing (P), and Sorting (S).

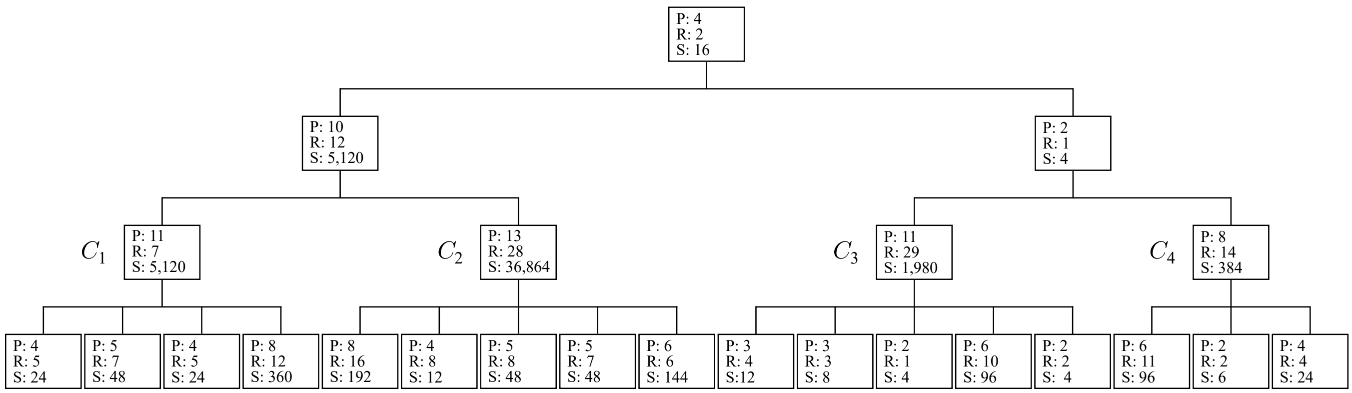

Fig. 8. Multilevel discrete-event control architecture, automatically derived from clustered DSM in Fig. 7. P denotes the number of component models, R the number of requirement models, and S the number of states in the supervised subsystem.

VI. Multilevel synthesis

The multilevel control structure has been derived automatically from Fig. 7. The top node corresponds to the module at the first level. Its children are the modules at the second level, and so on. The resulting structure is given in Fig. 8. The number of component and requirement models, as well as the number of states in the supervised subsystem are given in the figure. The total number of the component models on the fourth level equals the 77 components in the model; the number of requirements equals the 204 requirements in the model. The summation of the states in all subsystems is 50,368 states. The supervisors are synthesized using CIF 3 [19]. The implementation of the synthesis algorithm in CIF 3 follows the BDD-based approach of [22].

For this multilevel structure, the collection of supervisors obtained is nonconflicting, i.e., the synchronous composition is nonblocking. This can be established by applying the compositional nonblocking verification algorithm of [23]. Structuring the system, synthesizing the supervisors, and checking nonblockingness takes around 13 seconds on a HP ZBook laptop with Intel i7 2.40 GHz CPU and 8 GB RAM.

It is not always the case that the resulting collection of supervisors is nonconflicting. In such cases one may construct a coordinator that acts as an additional supervisor for eliminating the blocking situations. Such a coordinator can be found by applying monolithic synthesis on all the component and requirement models and all supervisors obtained, or by applying the compositional coordinator construction proposed in [24].

To compare the used technique to monolithic synthesis and multilevel synthesis with manual structuring, two more solutions have been analyzed. For a modeler to manually divide the production line in modules, it seems intuitive to choose a module for each individual station. Therefore, it is assumed that for the manually structured case each station belongs to a single module, i.e., instead of C_1, C_2, C_3, and C_4, there are now six modules at this level, and one module at the top level. The summation of the number of states in all the subsystems, the computation times, and the number of supervisors, are given in Table I.

TABLE I
Results of different synthesis approaches.

Synthesis architecture	States	Time [s]	Supervisors
MLDES synthesis automatically structured	5.0×10^4	13	44
MLDES synthesis manually structured	1.4×10^8	2	7
Monolithic synthesis	2.2×10^{25}	370	1

As can be seen, the multilevel supervisors have a smaller state representation than the monolithic supervisor. It also takes considerably less time to compute the multilevel supervisors. Furthermore, the automatically structured supervisors are smaller than the manually structured supervisors. Regarding computation time, the difference between the multilevel approaches is mainly due to the time it takes to start the clustering algorithm. However, the time it takes to manually identify the clusters is not taken into account here, which can be very time consuming. When using DSMs this identification step is automated.

VII. Code generation

As is mentioned in Section II, the supervisors of the production line are implemented on a PLC. PLCs do not support automata-based programming languages and do not have concepts such as event synchronization. The synthesized supervisors can therefore not be implemented directly. Here, we use the transformation described in [9] to remove synchronizations, such that the supervisors can be implemented in a sequential programming language. The resulting code scales linearly with the number of states and edges in the models. Since the synchronous product is not calculated explicitly, the number of states and edges remains small.

In CIF 3 [19], this transformation algorithm is implemented to generate structured text PLC code. Structured text adheres to international standard IEC-61131-3 for PLC code. It is a textual programming language. Code generation from automata models has successfully been used in previous projects, such as for baggage handling systems in [25]. For the production line, the generated code consists of 9200 lines of code and has a file size of 300kB.

The generated structured text PLC code was successfully installed on the PLC. The only manual activity required is the association of changes of the PLC input and output variables with uncontrollable and controllable events, respectively. Of course, our modeling of the plant has already assumed such a relationship. Typically, the rising edge is connected to an *on* event, whereas the falling edge is connected to an *off* event.

When observing the controlled behavior, some model mismatches were found, e.g., product sensors in the processing station switch on when the turntable is rotating over them. After improving the models, the new PLC code was generated in minutes (this includes altering the model, DSM-based structuring, and multilevel synthesis). After these improvements, the observed controlled behavior was in line with the behavior described in Section II. It should be noted that there could be differences in the behavior of the models, and the behavior of the code on a PLC, due to the nature of PLCs, e.g., simultaneity, and inexact synchronization [26]. However, we did not observe problems related to these phenomena. This can be explained by the short cycle time of the PLC (milliseconds) compared to the time of activities (seconds) in the production line.

The synthesized collection of supervisors seems suited for a distributed implementation on multiple PLCs. This needs to be investigated in detail.

VIII. Concluding Remarks

In this paper, a method for automatic generation of control architectures for supervisory control problems is illustrated in a production line case study. This method uses analysis techniques for DSMs to automatically derive the multilevel control architecture. It is shown that this approach leads to a considerable reduction in computational effort for synthesis compared to the monolithic approach. Moreover, it is shown that the resulting set of supervisors has successfully been implemented and validated on the actual hardware.

Acknowledgments

We thank Henk van Rooy for his assistance with the implementation aspects related to the Festo stations.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[2] N. C. W. M. Braspenning, J. M. van de Mortel-Fronczak, and J. E. Rooda, "A model-based integration and testing method to reduce system development effort," *ENTCS*, vol. 164, no. 4, pp. 13–28, 2006.

[3] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *MCSS*, vol. 1, no. 1, pp. 13–30, 1988.

[4] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Trans. Automat. Contr.*, vol. 35, no. 10, pp. 1125–1134, 1990.

[5] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Trans. Automat. Contr.*, vol. 37, no. 11, pp. 1692–1708, 1992.

[6] 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.

[7] M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, and J. E. Rooda, "Structuring multilevel discrete-event systems with dependency structure matrices," in *CDC*. IEEE, 2017, pp. 558–564.

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

[9] L. Swartjes, D. A. van Beek, and M. A. Reniers, "Towards the removal of synchronous behavior of events in automata," in *WODES*. Elsevier, 2014, pp. 188–194.

[10] W. M. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history–1980-2015," in *IFAC world congress*. Elsevier, 2017, pp. 1827–1833.

[11] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su, and J. E. Rooda, "Application of supervisory control theory to theme park vehicles," *DEDS*, vol. 22, no. 4, pp. 511–540, 2012.

[12] R. J. M. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda, "Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner," *IEEE Trans. Automat. Sci. Eng.*, vol. 11, no. 1, pp. 20–32, 2014.

[13] S. Adyanthaya, H. A. Ara, J. Bastos, A. R. B. Behrouzian, R. M. Sánchez, J. van Pinxten, B. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, M. Hendriks, S. Stuijk, M. A. Reniers, and J. Voeten, "xCPS: a tool to explore cyber physical systems," *SIGBED Review*, vol. 14, no. 1, pp. 81–95, 2016.

[14] J. Axelsson, A. Kobetski, Z. Ni, S. Zhang, and E. Johansson, "MOPED: A mobile open platform for experimental design of cyber-physical systems," in *SEAA*. IEEE, 2014, pp. 423–430.

[15] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[16] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Trans. Automat. Contr.*, vol. 51, no. 5, pp. 782–793, 2006.

[17] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. Automat. Sci. Eng.*, vol. 8, no. 3, pp. 560–569, 2011.

[18] S. Miremadi, K. Akesson, and B. Lennartson, "Extraction and representation of a supervisor using guards in extended finite automata," in *WODES*. IEEE, 2008, pp. 193–199.

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

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

[21] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, and J. E. Rooda, "CIF3 models for the small-scale production line," 2018, www.github.com/ffhreijnen/CCTAProductionLine.

[22] S. Miremadi and B. Lennartson, "Symbolic on-the-fly synthesis in supervisory control theory," *IEEE Trans. Contr. Syst. Technol.*, vol. 24, no. 5, pp. 1705–1716, 2016.

[23] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *DEDS*, vol. 26, no. 1, pp. 33–84, 2016.

[24] M. A. Goorden, M. A. Reniers, J. M. van de Mortel-Fronczak, W. J. Fokkink, and J. E. Rooda, "Compositional coordinator synthesis for blocking systems of extended finite automata," 2018, submitted for publication.

[25] L. Swartjes, D. A. van Beek, W. J. Fokkink, and J. A. W. M. van Eekelen, "Model-based design of supervisory controllers for baggage handling systems," *Simulation Modelling Practice and Theory*, vol. 78, pp. 28–50, 2017.

[26] J. Zaytoon and B. Riera, "Synthesis and implementation of logic controllers–a review," *Annual Reviews in Control*, 2017.