Supervisory Control Synthesis for a Waterway Lock

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

Abstract— **Formal methods help in coping with the growing functionality and complexity, time-to-market and costs in cyber-physical systems (CPSs). Supervisory control synthesis (SCS) is such a method. It can be used to synthesize a controller for a CPS from the uncontrolled system model (plant) and the specification model (requirements). While SCS is an active research topic, reports on industrial applications are rare. In this paper, we show the applicability of SCS to the design of controllers for waterway locks. The following steps in the development process are discussed: modeling the plant and the requirements, synthesizing the supervisor, validating the supervisor, generating a real-time controller and implementing this controller on a PLC. Following this way of working, a supervisory controller for a real waterway lock has been successfully developed and analyzed. The real-time controller is implemented in an experimental set-up. The state-space size of the uncontrolled plant is 6.0×10^{32} and the number of state-based requirements involved in the specification is 234. The synthesized controller is automatically translated into 1.2×10^4 lines of structured text, executable by a PLC. This case study delivers a proof of concept for the applicability of the procedure for supervisor synthesis and automatic PLC code generation to industrial-size systems.**

I. Introduction

Cyber-physical systems (CPSs) have become increasingly complex due to the high demands from the market in terms of functionality, quality and safety. As a result, controllers for these systems are getting more complex as well. At the same time, it is desired to decrease time-to-market and costs. Model-based development methods can help in overcoming these difficulties, as shown in [1] and [2]. The use of formal models has a vast advantage over the traditional engineering process, see e.g., [3][4]. Models can help in verifying and validating the controller design early in the process, resulting in a reduction of design errors found during the later testing and integration phase, where errors are more costly and time-consuming to repair.

This paper shows how formal methods can aid in the development of supervisory controllers. A supervisory controller is responsible for coordinating the activities in CPSs based on discrete observations of the system's state. These observations come from sensors measuring the state of the mechanical components in the system. Based on the observations, the supervisor decides which actions can safely be executed, and which actions have to be restricted. Actions typically involve the actuation of a mechanical component. Often, resource controllers are placed between the supervisor and the actuators and sensors for low-level control. This control structure is schematically depicted in Figure 1. Designing a supervisory controller is challenging, due to the large number of states the system can be in, leading to a high complexity in CPSs.

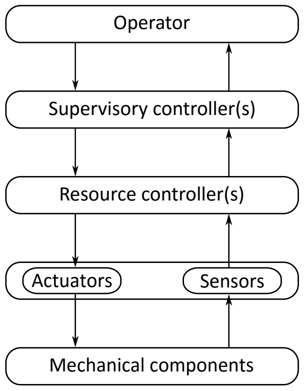

Fig. 1. Control structure in a cyber-physical system.

Supervisory control synthesis (SCS) [5] is a method for the design of supervisors. Formal models of the uncontrolled system (plant), and the specification (requirements) are used to synthesize a model of the supervisor. The supervisor is guaranteed to behave according to the specification, is non-blocking, controllable and maximally permissive. The synthesis procedure eliminates the manual design process of creating a supervisory controller that satisfies the requirements. Hence, the focus shifts from developing and debugging the implementation code to designing and debugging the requirements, which is often more intuitive. The supervisor model can be analyzed by verification tools and be validated by means of simulation. This increases the confidence in its correctness before implementation. Moreover, the supervisor model can be used for automatic generation of the implementation code.

Although SCS has widely been accepted within academia, reports on industrial applications are scarce. Previous successful attempts include control of a patient support table for an MRI scanner [6], a theme park vehicle [7] and a mobile robot [8]. Causes for the low number of industrial applications can be due to the large state-spaces often encountered in there. A second cause is the lack of commercially available tooling for SCS.

In this paper, a waterway lock has been investigated as an industrial application for SCS. It is shown that SCS is applicable to the design of supervisory controllers for lock systems. In contrast to the before mentioned applications, the uncontrolled state-space size of the lock is much larger (6.0×10^{32} compared to 5.2×10^4 for the theme park vehicle) and more requirements (234 compared to 34) are involved. Moreover, PLC code is automatically generated from the synthesized model of the supervisor and tested in an experimental set-up.

This paper is structured as follows. The waterway lock analyzed in this case-study is described in Section II. The formal plant and requirement models developed for the lock are discussed in Section III. The synthesis of the supervisor is discussed in Sections IV. Section V describes the validation and simulation of the synthesized supervisor. The automatic generation and implementation of the supervisor in a real-time experimental set-up is presented in Section VI. Finally, Section VII concludes the paper and discusses further research.

II. System description

A waterway lock is used in canals and rivers to facilitate raising and lowering of vessels between stretches of water of different levels. To this end, a chamber is used, that is separated from the rest of the canal by gates at the upper and lower end. In the chamber, the water level can be varied. An example of a lock with three chambers can be seen in Figure 2.

Fig. 2. Lock system in Maasbracht, the Netherlands.

For this case study, Lock III in Tilburg, the Netherlands has been considered. Lock III consists of only one chamber. A schematic representation of Lock III is shown in Figure 3. Various subsystems can be distinguished in Lock III: gates, paddles, culverts, two-lamp traffic lights and three-lamp traffic lights. Most other locks consist of similar subsystems. The gates are watertight doors which seal off the chamber from the outside water. Each gate can be opened or closed by a hydraulic actuating system. At the downstream side of the lock, paddles are built into the gates for emptying the chamber. At the upstream side of the lock, culverts can be opened to fill the chamber.

Traffic at both sides of the lock is regulated by traffic lights. At each entrance two traffic lights are positioned to regulate incoming vessels. Additionally, at each exit two traffic lights are used to regulate outgoing vessels. The outgoing traffic lights can display a red or a green aspect, signaling vessels to wait or to leave the lock, respectively. The incoming traffic lights can display a red, a red-green, a green or a red-red aspect. A red or a green aspect signals vessels to wait or to enter the lock, respectively. A red-green aspect indicates that the vessels can soon enter the lock, and a red-red aspect indicates that the lock is out-of-service.

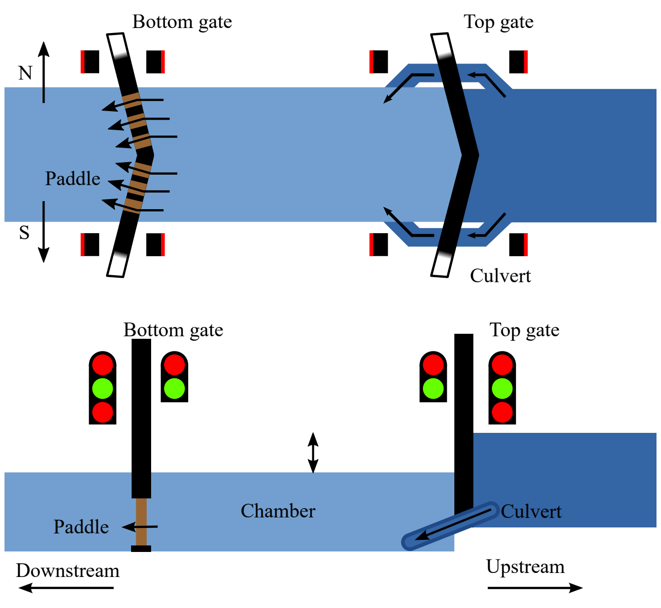

Fig. 3. A schematic representation of the top view (top) and side view (bottom) of Lock III.

Lock III is controlled from a remote control center, where operators monitor the lock via camera images. An operator controls the locks through a graphical user interface (GUI) implemented on a SCADA system. The GUI contains buttons to operate gates, paddles, culverts and traffic lights. For these components, commands to open, close and stop or to change a sign aspect, as well as an emergency stop are available. In total 25 different commands are available. Furthermore, graphically represented feedback from sensor signals is given via the GUI to the operator, e.g., the positions of the gates and valves, the sign aspects and the water heights.

III. Model development

In order to synthesize a supervisory controller for Lock III, the uncontrolled system and the control specifications are modeled. The behavior of the uncontrolled system is modeled using automata, as is common in the context of SCS. An automaton consists of states and transitions between them, which are labeled by events. Events can be either controllable or uncontrollable. Controllable events represent actions that can be influenced by the supervisor, e.g., enabling or disabling an actuator. Uncontrollable events cannot be restricted, e.g., a sensor switching on or off. Graphically, states are denoted by (labeled) circles, where an unconnected incoming arrow indicates the initial state, and filled circles indicate the marked state. A marked state denotes a safe state in the system. Transitions, labeled by controllable and uncontrollable events are depicted by solid and dashed arrows, respectively.

In this context, models of the control specification are either represented by automata or by state-based expressions. Here, the textual specifications of Lock III are intuitively translated into state-based expressions, corresponding to the syntax proposed in [9] and [10]. These state-based expressions come in two forms, x **needs** Y and Y **disables** x. The first means: event (set) x is restricted in all states, except for state(s) Y. The second means: event (set) x is restricted in all state(s) Y. These expressions are complementary in the sense that x **needs** $Y \Leftrightarrow \neg Y$ **disables** x. Depending on the size of Y and $\neg Y$, one of the two can be more convenient to use. Here, Y is in the form of a logical expression over multiple states, where $\neg$, $\vee$ and $\wedge$ are used for 'not', 'or' and 'and', respectively.

A. *Plant model*

In this subsection, the model of the uncontrolled behavior of Lock III is presented in detail[1]. The system model is decomposed into subsystems, which are further decomposed into components. Typically, components are either actuators or sensors. The behavior of the lock system is obtained by the parallel composition (synchronous product) of all subsystem components. This decomposition allows for small models at component level. The model decomposition for Lock III, excluding the GUI buttons is depicted in Figure 4. An arrow indicates the relation between the system and a subsystem, and between a subsystem and its components.

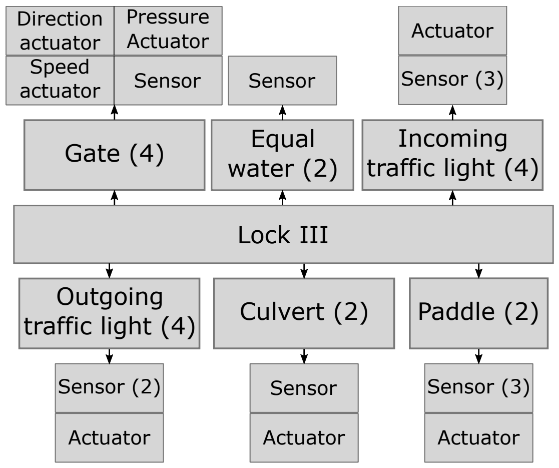

Fig. 4. A schematic decomposition of the plant, excluding the GUI. A number indicates the number of subsystems or components present.

The plant consists of the following subsystems: 4 outgoing traffic lights, 4 incoming traffic lights, 2 gate paddles, 2 culverts, 4 gates, 2 equal-water sensors and 25 GUI buttons. The subsystems have a unique name, depending on their position in the lock, 'D' or 'U' for downstream or upstream side, respectively, and 'N' or 'S' for north or south side, respectively. E.g., gate_DN refers to the north gate at the downstream side of the lock (see Figure 3). Subsystems are further decomposed into actuators '.A' and sensors '.S'. Below, the defined subsystems are explained in more detail.

1) Outgoing traffic light: The outgoing traffic light model is decomposed into an actuator and two sensors. The actuator is modeled as two states, red and green, which are the two defined aspects the traffic light can show. Initially, the red aspect is shown, as this is a safe state. The automaton depicted in Figure 5 contains this behavior.

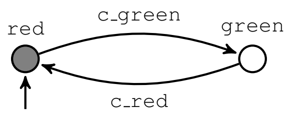

Fig. 5. Model of the outgoing traffic light actuator.

Two sensors provide feedback on the state of the lamps, e.g., the lamp is enabled or the lamp is disabled. Such a sensor is modeled with two states, `on` and `off`. The initial state for the red and green lamp sensors are `on` and `off`, respectively. The models for these sensors are depicted in Figure 6 by the left and right automaton, respectively.

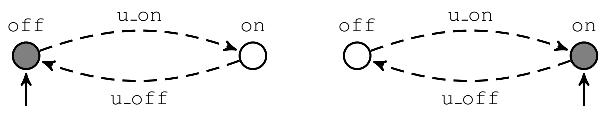

Fig. 6. Model of the outgoing traffic light green lamp sensor (left) and red lamp sensor (right).

2) Incoming traffic light: The incoming traffic light model is decomposed into an actuator and three sensors. The actuator can switch between the red, green, red-green and red-red aspects. Three requirements are already included in the model: the green, red-green and red-red states can only be reached from the red-green, red and red state, respectively. Figure 7 depicts the automaton that captures this behavior.

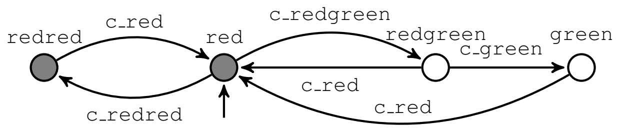

Fig. 7. Model of the incoming traffic light actuator.

The sensor model for the top red lamp is the same as the right automaton depicted in Figure 6, whereas the green and bottom red light sensors are equal to the left automaton depicted in Figure 6.

3) Gate paddle: The gate paddle model consists of an actuator that actuates a valve, controlling three cylinders connected to panels in the gate, and two end-position sensors per cylinder. The actuator can extend and retract the cylinders in order to close and open the panels, respectively. When not actuated, the cylinders are in a rest position, which is the initial position. This behavior is captured by the top automaton depicted in Figure 8.

Each cylinder is equipped with two sensors, measuring the fully retracted and fully extended positions. A state where both sensors are enabled simultaneously, does not exist physically, and is therefore not included in the model.

[1]The code is available at: github.com/ffhreijnen/LockCCTA

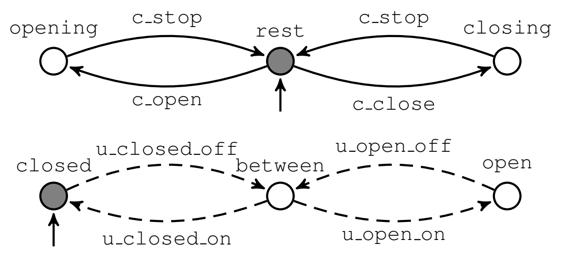

Fig. 8. Model of the paddle actuator (top) and sensor (bottom).

Initially, the paddles are closed. The bottom automaton in Figure 8 contains the sensor behavior.

4) Culvert: The culvert model consists of an actuator and a sensor. The actuator can be instructed to open or close the culvert (no rest position). Initially, the culvert is closed. Water flow through the culvert is measured with a flow sensor, where initially no flow is measured. This behavior is represented by the left and right automaton in Figure 9 for the actuator and sensor, respectively.

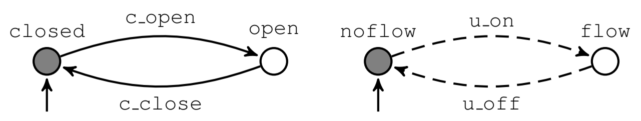

Fig. 9. Model of the culvert actuator (left) and sensor (right).

5) Gate: Controlling a gate is more involved. A gate can open (or close) by retracting (or extending) a hydraulic cylinder. When not actuated the cylinder is in a rest position. The automaton in Figure 10 represents the valve that controls this cylinder.

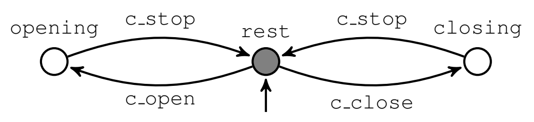

Fig. 10. Model of the gate direction actuator.

The pressure and movement speed in this cylinder are controlled to influence the movement of the gate. The pressure in the cylinder can be set to high or low for movement, and to off if the gate is in rest. The automaton in Figure 11 depicts this behavior.

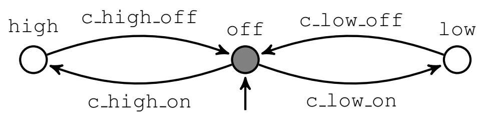

Fig. 11. Model of the gate pressure actuator.

Movement speed of the gate can be fast or slow. Fast is used for the initial movement and slow for the last few degrees of movement, preventing the gate from hitting the wall at full speed. The speed values are different when opening or closing, due to differences in water resistance. In total four speeds settings are available and a rest position. It is not possible to choose more than one setting. This behavior is represented by the automaton in Figure 12.

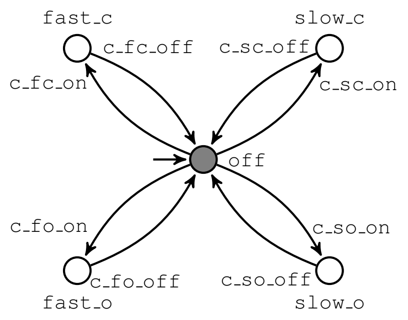

Fig. 12. Model of the gate speed actuator.

Additionally, six position sensors measure the extension of the cylinder. This is to differentiate between the two end positions, the acceleration points and the break points. The automaton given in Figure 13 shows all possible sensor positions. Positions 'bclosed' and 'bopen' indicate the points where the speed should change from fast to slow. Similarly, positions 'fclosed' and 'fopen' indicate where the pressure should be decreased. Fast speed and high pressure should be chosen when the gate starts opening or closing.

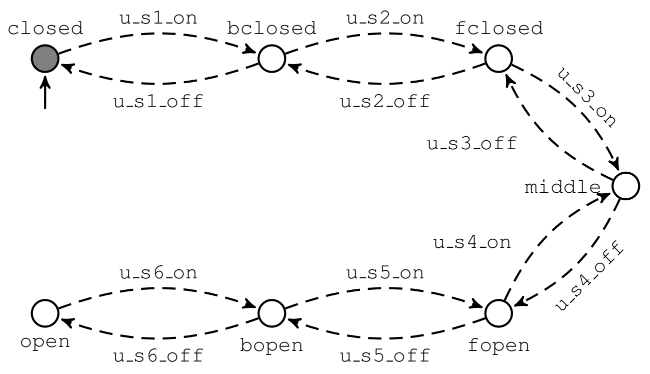

Fig. 13. Model of the cylinder extension sensors.

6) Equal-water sensor: Two analog water-height sensors measure the difference in water height over a gate. These two sensors are modeled together as one equal-water sensor. That is, the equal-water sensor is on if the water height measurements are equal, within a certain error margin. This behavior is modeled as an on-off sensor.

7) Graphical user interface: The graphical user interface consists of

- 20 buttons to operate the lock in normal conditions,
- 4 buttons to stop the movement of the gates, paddles or culverts, and
- an emergency button that immediately stops all movements.

Every button is modeled as an automaton with two states, the state where the button is pushed and the state where the button is released. Initially, all buttons are released. Figure 14 shows the two-state automaton.

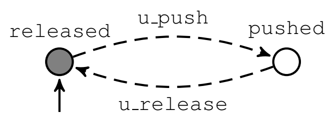

Fig. 14. Model of the GUI button.

B. Requirement models

There are several requirements imposed on the lock to guarantee its safe and correct behavior. The list of safety requirements for the lock system is given below. The state-based expression for the actuators at the upstream side of the lock are given below each requirement. The naming convention of the states in the requirements is as follows: <subsystem>.<component>.<location>, referring to a location in a plant model.

1) A paddle or a culvert is not allowed to open, see Figure 15, if:
   a) a paddle/culvert at the opposite side is not closed
   b) a gate at the opposite side is not closed

```
¬pad_DN.S1.closed ∨ ¬pad_DN.S2.closed ∨
¬pad_DN.S3.closed ∨ pad_DN.A.opening  ∨
¬pad_DS.S1.closed ∨ ¬pad_DS.S2.closed ∨
¬pad_DS.S3.closed ∨ pad_DS.A.opening  ∨
¬gate_DN.S.closed ∨ gate_DN.A.opening ∨
¬gate_DS.S.closed ∨ gate_DS.A.opening
disables {culvert_UN.c_open, culvert_US.c_open}
```

Fig. 15. Model of requirement 1: opening the upstream culverts.

2) A gate is not allowed to open, see Figure 16, if:
   a) there is no equal water over the gate
   b) a paddle/culvert at the opposite side is not closed
   c) a gate at the opposite side is not closed

```
    equal_U.S.off ∨
¬pad_DN.S1.closed ∨ ¬pad_DN.S2.closed ∨
¬pad_DN.S3.closed ∨ pad_DN.A.opening  ∨
¬pad_DS.S1.closed ∨ ¬pad_DS.S2.closed ∨
¬pad_DS.S3.closed ∨ pad_DS.A.opening  ∨
¬gate_DN.S.closed ∨ gate_DN.A.opening ∨
¬gate_DS.S.closed ∨ gate_DS.A.opening
disables {gate_UN.c_open, gate_US.c_open}
```

Fig. 16. Model of requirement 2: opening the upstream gates.

3) A gate is not allowed to close, see Figure 17, if:
   a) an incoming traffic light does not display the red or red-red aspect
   b) an outgoing traffic light does not display the red aspect

```
¬(in_UN.S.red ∨ in_UN.S.redred) ∨
¬(in_UN.A.red ∨ in_UN.A.redred) ∨
¬(in_US.S.red ∨ in_US.S.redred) ∨
¬(in_US.A.red ∨ in_US.S.redred) ∨
¬out_UN.S.red ∨ ¬out_UN.A.red   ∨
¬out_US.S.red ∨ ¬out_US.A.red
disables {gate_UN.c_close, gate_US.c_close}
```

Fig. 17. Model of requirement 3: closing the upstream gates.

4) An incoming traffic light is not allowed to display the green aspect, see Figure 18, if:
   a) an outgoing traffic light displays the green aspect
   b) the gates at the same side are not completely open

```
 out_UN.S.green ∨ out_UN.A.green    ∨
 out_US.S.green ∨ out_US.A.green    ∨
¬gate_UN.S.open ∨ gate_UN.A.closing ∨
¬gate_US.S.open ∨ gate_US.A.closing
disables {in_UN.c_green, in_US.c_green}
```

Fig. 18. Model of requirement 4: enabling the green aspect for the upstream incoming traffic lights.

5) An outgoing traffic light is not allowed to display the green aspect, see Figure 19, if:
   a) an incoming traffic light displays the green aspect
   b) the gates at the same side are not completely open

```
  in_UN.S.green ∨ in_UN.A.green     ∨
  in_US.S.green ∨ in_US.A.green     ∨
¬gate_UN.S.open ∨ gate_UN.A.closing ∨
¬gate_US.S.open ∨ gate_US.A.closing
disables {out_UN.c_green, out_US.c_green}
```

Fig. 19. Model of requirement 5: enabling the green aspect for the upstream outgoing traffic lights.

Aside from the safety requirements, there are additional requirements that add functionality to the GUI buttons and requirements for enabling and disabling certain actuators in te gates.

The function of the GUI buttons is to enable an actuator when the corresponding button is pushed. These requirements can again be modeled using state-based expressions, like: '`X.act.c_on` **needs** `button.pushed`', where `X` is the relevant subsystem. The following actions are restricted via the GUI buttons: switching traffic light aspects and opening and closing of the gates, culverts and paddles.

Another type of requirement is stopping an actuator when its end position is reached, or when the stop command is given. These requirements are modeled similar to the GUI requirements, e.g., '`X.act.c_stop` **needs** `X.sen.closed ∨ stop.pushed`', where `X` is the associated subsystem.

Finally, when opening or closing a gate, the corresponding pressure and speed valves need to be controlled (direction valve is controlled via GUI). Table I shows when to activate or deactivate the pressure and speed actuators during closing of the gate, depending on the sensor state (Figure 13). These requirements can be modeled straightforward from the table. For instance, the activation of the slow_c speed actuator can be modeled by: '`X.actSpe.c_sc_on` **`needs`** `X.sen.bclosed` $\wedge$ `X.actDir.closing`', where `X` is a gate subsystem.

TABLE I
REQUIREMENTS FOR CORRECT BEHAVIOR OF THE GATE WHILE CLOSING.

	Pressure		Speed			
Sensor	high	low	fast_o	slow_o	fast_c	slow_c
open	1	0	0	0	1	0
bopen	1	0	0	0	1	0
fopen	1	0	0	0	1	0
middle	1	0	0	0	1	0
fclosed	0	1	0	0	1	0
bclosed	0	1	0	0	0	1
closed	0	0	0	0	0	0

IV. Synthesis of the Controller

A supervisory controller has been synthesized from the plant and requirement models, specified in Section III. The models are implemented in the CIF toolset [11], which supports automata and state-based expression models. The synthesis algorithm implemented in CIF is based on the algorithm proposed in [12]. This algorithm uses binary decision diagrams (BDDs) to efficiently store and do computations even for large state-space models. The computations are done on a network of automata and state-based expressions, without unfolding the complete state-space. Hence, this algorithm allows for the synthesis of supervisory controllers for industrial-size problems.

The uncontrolled state-space size of the plant, is 6.0×10^{32}. In the control specification, 234 state-based requirements are involved. Synthesizing the supervisor takes a few seconds on an i7, 2.60GHz, 8GB laptop. The resulting network of automata represents a supervisor containing 6.0×10^{24} states, which is a reduction of a factor 10^8.

The synthesized supervisor coordinates the system such that it is guaranteed to satisfy the following properties.

- Safety: the system cannot reach states that are forbidden by the requirements.
- Controllability: the supervisor only restricts controllable events, if needed.
- Non-blockingness: the supervisor does not prevent the system from reaching a marked state.
- Maximal permissiveness: the supervisor imposes the minimal restriction on the plant to satisfy safety, controllability and non-blockingness.

V. Simulation of the Controlled System

Although the system is guaranteed to behave according to the requirements, the resulting closed-loop behavior might not be as expected. For example, requirements could be too strict and as a results, the supervisor could prevent a part of the desired behavior. Another cause for unexpected behavior can be due to inadequate modeling. Hence, the resulting supervisor has to be validated. Simulation is used to validate whether the model of the controlled system is consistent with the intended behavior.

The CIF simulator supports real-time, interactive simulation and animation, by combining the controlled system with an image in the Scalable Vector Graphics (SVG) format, as shown in Figure 20. The state of the lock is visualized on the left hand side. On the right hand side, an interactive control panel is placed, which is used to simulate commands given by the lock operator via the GUI.

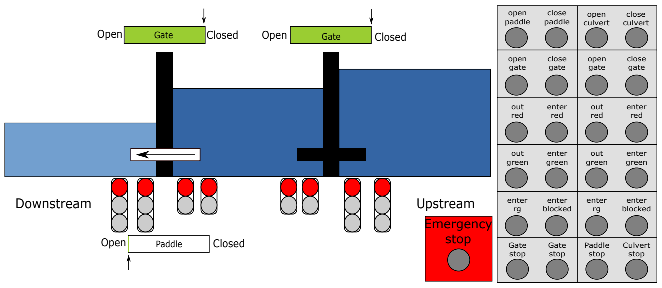

Fig. 20. SVG image for simulation-based visualization.

A hybrid uncontrolled system model (hybrid plant) is created for the simulation. The hybrid plant is obtained by extending the discrete-event plant model with additional continuous behavior, e.g., a model of the behavior of the water inside the lock and timing information for cylinder movements. This allows for animation in the simulation.

Normal operations as well as illegal operation of the lock are simulated. All desired behavior, i.e., facilitating raising and lowering vessels between the different water levels, is present. The illegal operation, i.e. giving unsafe operating commands, did not influence the system's state. This is as expected, as these illegal operations are all restricted by the safety requirements.

VI. Implementation of the Controller

Supervisory controllers in industry are often implemented on a programmable logic controller (PLC), as is the case for the lock system. A downside of PLCs is that they only support sequential programming languages and do not allow event synchronization. This makes it impossible to directly implement the synthesized supervisor as a network of automata. In order to implement the supervisor, all synchronizations have to be removed.

A straightforward solution is to unfold the whole controlled state space, containing all behavior of the system in a single automaton without synchronization. However, most often the state space becomes too large to compute, here 6.0×10^{24}. In [13], an algorithm is described that removes synchronization while avoiding a state-space explosion. The elimination of synchronization is accomplished by normalizing states and events and removing synchronous behavior. Nevertheless, the behavior of the original and the adapted supervisor is the same. The final result does not contain events that synchronize different automata, making implementation in a sequential programming language possible.

In CIF, a code generation algorithm [14] is implemented to generate structured text (ST) PLC code that adheres to the IEC 61131-3 industrial standard. For the supervisor of Lock III this generation results in 1.2×10^4 lines of ST.

While Lock III is still under construction, testing the generated PLC code on the hardware is not yet possible. Therefore, an experimental set-up has been used to check for the correct implementation of the supervisor, see Figure 21. This set-up is a simplified model of Lock III, but it still includes all essential functions described in Section II. Thus, the subsystems modeled in Section III are all present in the set-up. The main differences between the set-up and Lock III are:

- The gate subsystem consists of a single direction actuator and two end-position sensors.
- There are no traffic light feedback sensors.
- There is no differentiation between north and south subsystems.

These differences are only of minor importance. Therefore, the set-up is representative for the lock III behavior.

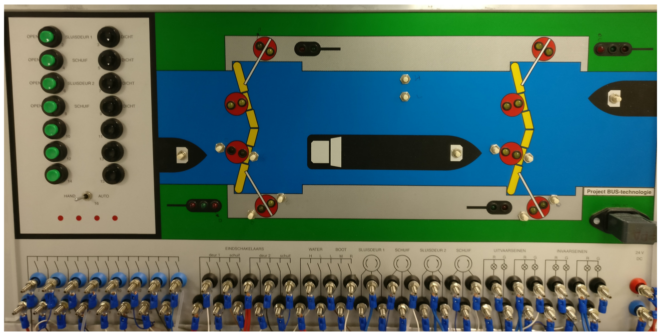

Fig. 21. Experimental set-up, simplified version of Lock III.

The set-up consists of a display where the lock is visualized, and a control interface with 14 buttons. The actuators, sensors and buttons are connected to 44 I/O contacts. The I/O contacts are connected to a TwinCAT softPLC from Beckhoff. A laptop is connected to the softPLC to run ST code. The ST code for the set-up is generated from the supervisor synthesized from a subset of plant and requirement models defined in Section III. The result contains 4,900 lines of code. Implementation of the generated code on the PLC is straightforward, as it can be uploaded without alterations. The only manual process is assigning the events to the corresponding I/O contacts.

While testing the behavior of the closed-loop system, no anomalies were observed. The behavior matches the exact behavior as in the simulation. This is in line with the claim that errors are found earlier in the design process and a lot less during the testing and integration phase.

VII. Concluding remarks

The results achieved in this case study deliver a proof of concept for the applicability of the procedure for supervisor synthesis and automatic PLC code generation for industrial-size systems. It has been shown that the current tooling can deal with the large state space of this industrial-size system, both when synthesizing and generating PLC code.

Future work will focus on investigating if SCS can be incorporated in the existing design process for locks. To this end, implementation and testing of the generated PLC code on the lock system hardware has to be considered.

Acknowledgments

We would like to thank Rijkswaterstaat, part of the Dutch Ministry of Infrastructure and the Environment, which has provided funding for this research. In particular we would like to thank Han Vogel, Maria Angenent, Robert de Roos and John van Dinther for their enthusiastic involvement in this project. We thank Hubert Hoen from Zuyd Hogeschool in Heerlen, the Netherlands, for providing the experimental set-up.

References

[1] J. Fitzgerald, P. G. Larsen, and J. Woodcock, "Foundations for model-based engineering of systems of systems," in *Complex Systems Design & Management*. Springer, 2014, pp. 1–19.

[2] J. C. M. Baeten, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Integration of supervisory control synthesis in model-based systems engineering," in *Complex Systems*. Springer, 2016, pp. 39–58.

[3] A. P. van der Meer, R. Kherrazi, and M. Hamilton, "Using formal specifications to support model based testing ASDSpec: a tool combining the best of two techniques," *arXiv preprint arXiv:1403.7257*, 2014.

[4] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM computing surveys (CSUR)*, vol. 41, no. 4, p. 19, 2009.

[5] 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.

[6] 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 Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 20–32, 2014.

[7] 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," *Discrete Event Dynamic Systems*, vol. 22, no. 4, pp. 511–540, 2012.

[8] C. R. C. Torrico, A. B. Leal, and A. T. Y. Watanabe, "Modeling and supervisory control of mobile robots: A case of a sumo robot," *IFAC-PapersOnLine*, vol. 49, no. 32, pp. 240–245, 2016.

[9] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782–793, 2006.

[10] J. Markovski, D. A. van Beek, R. J. M. Theunissen, K. G. M. Jacobs, and J. E. Rooda, "A state-based framework for supervisory control synthesis and verification," in *Conference on Decision and Control*. IEEE, 2010, pp. 3481–3486.

[11] 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 *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 575–580.

[12] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Engineering Practice*, vol. 14, no. 10, pp. 1157–1167, 2006.

[13] L. Swartjes, D. A. van Beek, and M. A. Reniers, "Towards the removal of synchronous behavior of events in automata," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 188–194, 2014.

[14] CIF. (2017) PLC code generation. [Online]. Available: http://cif.se.wtb.tue.nl/tools/cif2plc/index.html.