Lessons learned in the application of formal methods to the design of a storm surge barrier control system *

Martijn Goorden [*,**] **Joanna van de Mortel-Fronczak** [***]
Koen van Eldik [**] **Wan Fokkink** [***] **Jacobus Rooda** [***]

[*] *Department of Computer Science, Aalborg University, Aalborg, Denmark, (e-mail: mgoorden@cs.aau.dk).*
[**] *Rijkswaterstaat, Dutch Ministry of Infrastructure and Water Management, Utrecht, the Netherlands, (e-mail: martijn.goorden@rws.nl, koen.van.eldik@rws.nl)*
[***] *Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, the Netherlands, (e-mail: j.m.v.d.mortel@tue.nl, w.j.fokkink@tue.nl, j.e.rooda@tue.nl)*

Abstract: The Maeslantkering is a key flood defense infrastructural system in the Netherlands. This movable barrier protects the city and harbor of Rotterdam, without impacting ship traffic under normal circumstances. Its control system, which operates completely autonomously, must be guaranteed to work correctly even under extreme weather conditions, although it closes only sporadically. During its development in the 1990's, the formal methods Z and Spin were used to increase reliability. As the availability of industrial expert knowledge on these formal methods declines, maintaining the specifications defined back then has become cumbersome. In the quest for an alternative mathematically rigorous approach, this paper reports on an experience in applying supervisory control synthesis. This formal method was recently applied successfully to other types of infrastructural systems like waterway locks, bridges, and tunnels, with the purpose to ensure safe behavior by coordinating hardware components. Here, we show that it can also be used to coordinate several (controller) software systems. Additionally, we compare the lessons learned from the originally used formal methods and link Z to supervisory control synthesis.

1. INTRODUCTION

In the Netherlands, a large part of the country is situated below sea level. The country is protected by several flood defense systems. One of them is the movable storm surge barrier called the Maeslantkering, located near Rotterdam, which was designed and constructed in the 1990s. Under normal circumstances, the barrier is open so that ships have access to the port of Rotterdam. Only under extreme weather conditions the barrier will close to protect the Rotterdam metropolitan area. The Maeslantkering is being managed by Rijkswaterstaat, the executive branch of the Dutch Ministry of Infrastructure and Water Management that is responsible for commissioning and maintaining major infrastructural systems in the Netherlands. The Maeslantkering is considered to be a major safety-critical system, as a failure to close (or open) can result in devastating water damage. Therefore it operates fully autonomously, thereby eliminating human errors in decision making. Its control system, called the 'Beslis en Ondersteunend Systeem' (Decision and Support System, abbreviated to BOS), decides when and how to close the barrier based on weather forecasts, water level measurements, and results from flood simulations.

The BOS has been classified to the highest Safety Integrity Level (SIL) 4 of IEC 61508 (IEC, 2010). For its design, formal methods were employed (Kars, 1996, 1998). The Z notation (Spivey, 1992) was used to formalize the functional view of the BOS (which functions and data structures exist) and Promela (Holzmann, 1991) to formalize the behavioral view (in which order should functions be executed). Promela models are verified using the Spin model checker (Holzmann, 1997). In Tretmans et al. (2001), the authors concluded that the use of formal methods helped to improve the quality of the BOS.

While the current BOS implementation is still functioning as intended, the availability of industrial expert knowledge on the formal methods used for developing it declines. Therefore it is questionable whether an existing software company would be able to re-use existing specifications in a renovatation of the current BOS implementation or the development of a new control system. Therefore, Rijkswaterstaat is exploring alternative formal methods for the specification and design of the BOS.

Supervisory control theory of Ramadge and Wonham (1987, 1989) provides means to synthesize supervisors from a model of the uncontrolled plant and a model of the

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

Fig. 1. The Maeslantkering in its closed position. Image from `https://beeldbank.rws.nl`, Rijkswaterstaat.

control requirements. Synthesis guarantees by construction that the closed-loop behavior of the supervisor and the plant adheres to all requirements, is nonblocking, is controllable, and is maximally permissive.

Recently, supervisor synthesis has been applied to design supervisors of different infrastructural systems: waterway locks (Reijnen et al., 2017), movable bridges (Reijnen et al., 2020), and road tunnels (Moormann et al., 2020). An overview of those case studies can be found in Goorden et al. (2020). They show that supervisor synthesis is very suitable to derive controllers that can be implemented on Programmable Logic Controller (PLC) hardware. Compared to those three case studies, the BOS is one level higher in a typical control stack: it communicates with and supervises local supervisory controllers at the Maeslantkering, the Hartelkering, and the Hartelsluis, where the latter two are other systems part of the same flood protection infrastructure near Rotterdam. Furthermore, it interacts with several other software systems, such as the aforementioned flood simulation environment.

In this paper, we discuss our experience with applying supervisory control synthesis to the design of the BOS. In this context, component models of the plant represent underlying software systems. We describe the BOS in some detail and identify which parts of it are suitable for supervisor synthesis. The Z specifications of these parts were translated into automata. A challenge was to cast the high-level Z specifications into automata that represent concrete system executions. To give an impression of the translations, we show some examples of how plants and requirements are modeled, and relate them to Z schemas used in the original specification. Finally, we discuss the lessons learned from applying supervisory control synthesis in comparison to the originally used formal methods.

This paper is structured as follows. Section 2 provides a description of the Maeslantkering and the BOS. Section 3 introduces theoretical preliminaries for this paper. In Section 4, we discuss and illustrate the modeling of the BOS in the context of supervisory control synthesis. Section 5 compares the two approaches: supervisory control synthesis versus formal methods using the Z notation and Promela as specification languages. Section 6 concludes the paper.

2. SYSTEM DESCRIPTION

Fig. 1 shows a picture of the Maeslantkering in its closed position, preventing the higher water coming from the sea (at the bottom of the picture) from entering the port of Rotterdam (at the top of the picture). Several high-level requirements were formulated for this movable storm surge barrier (Tretmans et al., 2001): Rotterdam should be protected from flooding, the port should always be reachable (except under extreme weather conditions), water coming from the river Rhine should not cause a landside flooding, and aquatic life should not be disturbed.

The Maeslantkering consists of two movable and hollow doors, each having a height of 22 meters and a length of 210 meters. These doors are connected by steel arms to pivot points on the banks, making each arm as large as the Eiffel Tower. During normal weather conditions, the Maeslantkering is open and the doors rest in their docks. Only when storms are expected with a danger of flooding, the doors are closed. Closing the doors is performed in several steps (Kars, 1998; Tretmans et al., 2001). First the (dry) docks are filled with water to let the hollow arms float. Subsequently, the doors are moved to the middle of the rivers. Finally, the doors are partially filled with water, making them sink to the bottom and closing the river.

Reliability of the system is key. The failure probabilities of unjustified not closing and not opening should be below 10^{-5}, see Kars (1998); Tretmans et al. (2001). Careful analysis during its design has shown that human decision making in critical situations undermines the reliability. Therefore, the decision when to close the barrier and the operation of the closure are fully automated with the control system BOS. These decisions are sent to local supervisory controllers at the Maeslantkering and two other nearby infrastructural objects.

The BOS consists of several (software) subsystems, each classified as core, critical, or non-critical, see Tretmans et al. (2001). Some of these subsystems are the internal communications library, the external communications library, the hydraulic flood simulation model interface, the database, the graphical user interface, and the BOS script. The last module contains all decision rules on when and how to operate the Maeslantkering. To evaluate these rules, the BOS collects data from different weather and water sensor stations, monitors the current state of the Maeslantkering, and uses flood simulations to obtain water level forecasts.

Ensuring that the storm surge barrier meets the high reliability requirements is challenging. It is rarely used: each year there is a single test closure just before the storm season, and the last time it closed due to an actual storm was in January 2018. Still, when it is used, it should execute flawlessly, including proper fault detection and handling. This is fundamentally different from the other case studies using supervisor synthesis where the systems are more or less continuously in operation, thus exciting all different parts of the system regularly.

3. PRELIMINARIES

This section provides a brief introduction of automata and supervisory control synthesis. Moreover, a short explanation of Z schemas is given.

3.1 Automata

In supervisory control theory, a distinction is made between the plant model and the requirements model. The plant model describes the uncontrolled behavior of the system, while the requirements model captures the desired behavior.

In this paper, we model the plant with deterministic extended finite automata (EFAs) (Sköldstam et al., 2007). An EFA is a 7-tuple $(L, X, \Sigma, E, L_m, l_0, v_0)$ where L is a finite set of locations, X a finite set of bounded discrete variables, Σ a finite set of events, $E \subseteq L \times \mathcal{G} \times \Sigma \times \mathcal{U} \times L$ a finite set of edges with $\mathcal{G}$ the set of guard expressions over variables in X and locations in L, and $\mathcal{U}$ the set of update functions for variables in X, $L_m \subseteq L$ a set of marked locations, $l_0 \in L$ the initial location, and v_0 the initial valuation of the variables. We use **T** and **F** to denote the Boolean values true and false, respectively.

In an EFA, an edge $(l_1, g, \sigma, u, l_2) \in E$ is enabled if the current location is l_1 and the guard g evaluates to **T** for the current valuation. After taking the edge, the current location is l_2 and the valuation is updated according to u.

For the purpose of supervisory control synthesis, the event set σ is partitioned into controllable events Σ_c and uncontrollable events Σ_u. A supervisor can disable controllable events, such as an actuator switching on, while it cannot disable uncontrollable events, such as a sensor switching on.

Requirements can be modeled with either EFAs or event conditions. An event condition is an expression of the form σ **needs** c or c **disables** σ, where $\sigma \in \Sigma$ and $c \in \mathcal{G}$, see Markovski et al. (2010). The first expression indicates that an event σ is only allowed when the condition c evaluates to **T**, while the latter indicates that σ is not allowed when c evaluates to **T**. While always one event condition form can be rewritten into the other, having both helps to increase the understandability of the model.

For large-scale systems, the plant model is typically given as a set of EFAs $\mathcal{P} = \{P_1, \ldots, P_m\}$ and the requirements model as a set of EFAs, event conditions, or a combination of both $\mathcal{R} = \{R_1, \ldots, R_n\}$. An individual $P_i \in \mathcal{P}$ is called a component model; an individual $R_j \in \mathcal{R}$ is called a requirement model.

3.2 Supervisory control synthesis

The objective of supervisory control theory is to design an automaton called a supervisor which dynamically disables controllable events so that the closed-loop system of the plant obeys the specified requirements, see Ramadge and Wonham (1987, 1989); Cassandras and Lafortune (2008); Wonham and Cai (2018). The following properties should be satisfied.

- *Safety*: all possible behavior of the closed-loop system should always satisfy the imposed requirements.
- *Controllability*: uncontrollable events may never be disabled by the supervisor.
- *Nonblockingness*: the closed-loop system should be able to reach a marked state from every reachable state.

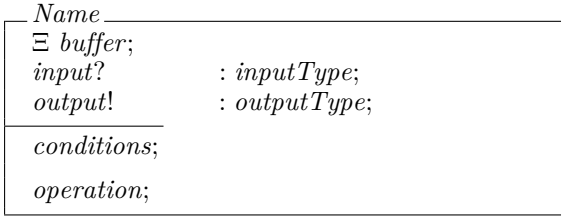

Fig. 2. Example Z schema.

- *Maximal permissiveness*: the supervisor does not restrict more behavior than strictly necessary to enforce safety, controllability, and nonblockingness.

Several tools exist that can calculate a supervisor for a given plant and requirements model. In this paper, CIF toolset (van Beek et al., 2014) is used, which has implemented monolithic synthesis of Ouedraogo et al. (2011) combined with binary decision diagrams, see Miremadi et al. (2011). Since 2020, CIF toolset is available and is being developed further as Eclipse Supervisory Control Engineering Toolkit (Eclipse ESCET™, ESCET (2022)).

3.3 Z schemas

Z (Spivey, 1992) is a formal specification language for modeling distributed computer systems. Its notations are based on set theory and predicate logic. A Z specification is divided into components called schemas, which specify both dynamic and static aspects of the system. Dynamic aspects include the transitions between states, while static aspects include allowed reachable states as well as invariants that must be preserved by transitions. In Fig. 2, a template of a typical Z schema is depicted. The symbol Ξ indicates that in the specified operation, the state of the buffer does not change. The question mark of the *input?* variable of type *inputType* indicates this is an input variable, while the exclamation mark of the *output!* variable of type *outputType* indicates this is an output variable. The *conditions* part is an abstract representation of conditions on the two variables, for instance that the input value must belong to a certain set. The *operation* part is an abstract representation of the operation that is performed, for instance which output value is returned.

4. MODELING FOR SUPERVISOR SYNTHESIS

As explained, the aim of supervisory control theory is to synthesize a supervisor that disables (controllable) events. Not all subsystems of the BOS are suitable for this theory to be applied, like, e.g., the communication libraries and the hydraulic flood simulation model interface. We only modeled those parts of the BOS script subsystem that are in line with the aim of supervisory control theory. In this section, we illustrate several modeling challenges with examples. Unfortunately, due to confidentiality, the full model consisting of EFAs cannot be made public. We only present some snippets in the form of finite automata.

4.1 Modeling the plant components

As mentioned in the system description in Sect. 2, the BOS coordinates lower level supervisors. For example, the BOS

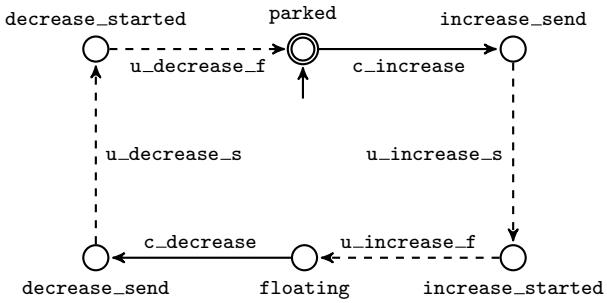

Fig. 3. The component model that captures the water leveling inside the docks, called Barrier.

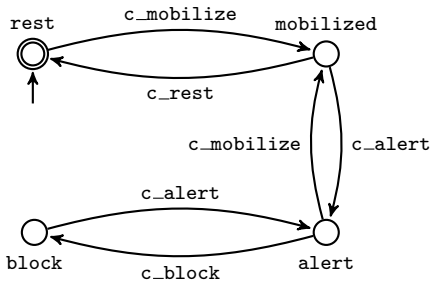

Fig. 4. The component model of the BOS script structure.

can request the local supervisor of the Maeslantkering to start increasing or decreasing the water level inside the docks. Subsequently, it gets a signal back when the lower level supervisor receives or finishes the command.

A challenge was to determine how the relevant part of the lower level supervisor should be abstracted for the BOS model. Fig. 3 shows the component model that represents the water leveling inside the docks. The BOS is able to send a command to the Maeslantkering to start increasing the water level inside the docks. After that, it will first receive a signal back that the command has been received and execution has been started, followed by a signal indicating that the process has finished. Decreasing the water level happens in a similar fashion.

From the documentation, it is unclear whether it is possible for the BOS to send a decrease water level command before the increasing process has finished completely, i.e., to allow transitions from locations `increase_send` and `increase_started` to `decrease_send` labeled with `c_decrease`. The model in Fig. 3 does not allow for this behavior, while one could also argue to include those transitions and formulate requirements to prevent this from happening.

To cope with the size of the BOS and the Maeslantkering, the specification of closure proceedings and the derived decision rules are structured into main phases, phases, and sub-phases. Each sub-phase is manageable by an engineer. Yet, supervisory control synthesis does not need such a structure. Still we could not ignore it, as numerous requirements are mentioning main phases, phases, or sub-phases, see also Sect. 4.2.

We modeled the given structure as a component model, albeit that it is not representing any physical or software subsystem. Fig. 4 shows a small part of the complete structure. In this part, the actual barriers of the Maeslantkering are moved from their docks into the middle of the river and vice versa, which is a relatively sequential process. Since this component is internal for the BOS script module, we classified all events as controllable. While a main phase can have several phases and a phase can have several sub-phases, not all main phases and phases have phases and sub-phases, respectively. Therefore, we decided to model the lowest elements in this decomposition and refer to, for example, '`alert` $\vee$ `block`' if we need to refer to the phase `alert`, which consists of the (modeled) sub-phases `alert` and `block`.

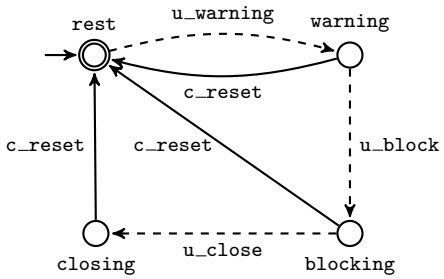

Fig. 5. The component model capturing the discretized time progress.

The third example to show is the modeling of time progression, if it can be expressed in discrete steps. Several decisions depend on the relative difference between the current time and the forecasted moment at which the water level limits will be exceeded. The period before a forecasted flood is divided into several discrete phases. Therefore, such timing aspects can be modeled with discrete intervals, which can be captured by a finite automaton.

Fig. 5 shows the model of the discretized time. Three specific time transitions are modeled with uncontrollable events. For instance, the state `warning` indicates that the current time is in between the time moments indicated by events `u_warning` and `u_block`. The BOS script is also able to reset the automaton when, for example, a new weather forecast no longer predicts extreme high water levels in the near future.

4.2 Modeling the requirements

While formal methods were used during the original design of the BOS, the BOS script was unfortunately not formalized, see Sect. 5 for a more in-depth discussion. Therefore, textual specifications of the decision rules have been translated into requirement models.

The major issue with formulating the requirements model is that a textual specification typically documents an intended solution path (as a sequential list of steps) and not a requirement that any solution path should satisfy. For example, in order to go to the `block` phase in the script structure model (see Fig. 4), a message has to be sent to the harbor control center first, then the water level should be increased, and then the dock doors should be opened. While this sequence can be modeled as a requirement expressed by the corresponding automaton, in general, explicitly specifying the solution with a requirement model is not the intention of supervisory control theory.

To translate a sequence of steps into a requirement model, we had to find the underlying conditions restricting the order of events, i.e., finding out *why* this is a desired order. This often required detailed discussions with BOS experts. Sometimes obtaining the underlying requirements was straightforward: for example, we modeled the requirement stating that the dock doors may only be opened when the water level is fully increased as

$$\texttt{Dock.c\_open}\ \textbf{needs}\ \texttt{Barrier.floating} \tag{1}$$

with `Dock.c_open` the event representing the command to open the dock doors (of which the model is not shown here) and `Barrier.floating` the state `floating` in Fig. 3. Yet, occasionally, identifying the underlying requirements was more complicated: for example, we first modeled the requirement stating that the water level inside the dock can only increase once the harbor control center has confirmed the message sent as

$$\texttt{Barrier.c\_increase}\ \textbf{needs}\ \texttt{MessageBlock.confirmed} \tag{2}$$

with `Barrier.c_increase` being the event in Fig. 3 and `MessageBlock.confirmed` the state in the (not shown here) model, where the block phase message from the BOS has been confirmed by the harbor control center. After discussions with experts, it turned out that the BOS is allowed to start increasing the water level even without receiving the confirmation from the harbor control center. When this confirmation is not received withing a certain time limit, the BOS should generate a warning and just continue. Therefore, the requirement in Equation 2 has been adjusted into

$$\texttt{Barrier.c\_increase}\ \textbf{needs}\ \texttt{MessageBlock.confirmed} \vee \texttt{MessageBlock.sent} \tag{3}$$

with `MessageBlock.sent` being the state where the block phase message has been sent by the BOS.

In total, the system is described with 19 component models and 95 requirement models. Using CIF toolset, we were able to synthesize a monolithic supervisor in a couple of seconds, where the supervisor is represented with a single EFA alongside the provided component models and requirement models, see Miremadi et al. (2008). The resulting controlled state space consists of approximately $8 \cdot 10^6$ states and $4 \cdot 10^7$ transitions.

4.3 Documenting the model

Rijkswaterstaat is experiencing now that there is insufficient knowledge on Z and Promela available at their industrial partners to fully understand the formal specifications of BOS written 30 years ago. Moreover, although automata modeling knowledge and experience in the context of supervisor synthesis is present at academic partners of Rijkswaterstaat, engineers in industry tend to lack this too. To facilitate communication between academics and industry, the automata model has been fully documented.

The format of this document is inspired by the Rosetta stone. Each part of the model, be it a component model or a requirement model, is described in three ways: plain text, mathematical notation, and CIF specification. Therefore, experts of the BOS that are not proficient in understanding automata or the CIF modeling language are still able to take part in discussing and evaluating the model.

The documentation also allows the modeler to report on modeling decisions, such that this knowledge is recorded and accessible for the future. This is an improvement compared to earlier practices, as previously modeling decisions typically remained undocumented.

5. FROM Z SCHEMAS TO AUTOMATA

When it comes to achieving safety guarantees for supervisors or logic controllers, two options exist, both anchored in formal methods. The first option, originating from the software engineering domain, consists in defining a formal model of the supervisor, which is then used to verify that the requirements are satisfied. Usually, this is done using model checkers which come with a specific modeling language. Generally speaking, for verification also the specification needs to be formalized. Both formalization steps are performed manually. The second option, originating from the control engineering domain, consists in defining a formal model of the system for which the supervisor is to be built and a formal model of specifications (functional and safety) that must be fulfilled. From these two models, the supervisor conforming to the specifications is synthesized, avoiding the need for verification. As in the first option, the associated two formalization steps are performed manually. Both options eventually result in the supervisor model which satisfies the specifications and from which control code can be generated.

At the time the control system of the Maeslantkering surge barrier was being developed, the first option was applied. As described in Tretmans et al. (2001), model checker Spin with its input language Promela (Holzmann, 1991, 1997) were used for verification of the design of three critical subsystems:

- monitoring and controlling other subsystems
- obtaining measurement data
- controlling the surge barrier PLC

The deadlock-free design of the complete system was achieved by following the rules of Martin and Welch (1997).

Additionally, for specifying (a part of) data and operations on them, the formal language Z was used. Using the ZTC type checker (Jia, 1995), correctness of syntax and types as well as completeness of the Z specifications were checked for all subsystems. Subsequently, the pre-conditions and post-conditions of the Z operation schemas were evaluated manually for all subsystems. Checking completeness of these schemas was also performed by manual inspection for core and critical subsystems. For the highly critical subsystems, the invariants were manually checked as well.

Back then, the control system code in a safe subset of C^{++} was not generated from the verified models but developed manually in a systematic and structured way based on them.

We performed a project with regard to the Maeslantkering in which a proof of concept was delivered for the applicability of the second option. To this end, two aspects were investigated. First, as described in Sect. 4, for a part

SendCommand

Ξ *dataTable*;
recCom? : *recComType*;
sendCom! : *sendComType*;

---

recCom?.*tn* $\in$ *tnSet* $\setminus$ {*valA*};
$\neg$(pre *pred*);

sendCom! = *exp*;

Fig. 6. The Z schema used to specify the function that sends a command to the Maeslantkering.

of one of the core subsystems, called the BOS script, a suitable model (without the controller) had to be derived. As this system has no direct interaction with actuators and sensors, the modeling experience from other infrastructural systems described in Reijnen et al. (2017, 2020); Moormann et al. (2020) could not be applied. From the available documentation, the associated specifications had to be derived. Second, we were asked by Rijkswaterstaat to construct automata models that are in line with the original specifications from 30 years ago. For this purpose, the relation between activities (events) present in the derived model and Z schemas had to be determined. Moreover, since Z schemas are at a high level of abstraction, we needed to make design decisions to turn abstract Z schemas into automata that represent concrete system behavior.

To illustrate the conversion from Z schemas to automata, an example Z schema, shown in Fig. 6, is considered which specifies sending commands to the PLC of the surge barrier. In this schema the elements are as follows.

- Input `recCom? : recComType` which indicates that the input type is `recComType`. Each element of this type consists of three values `(tn, tp, tt)` representing task name, task parameter, and the time at which the command was issued.
- Output `sendCom! : sendComType` which indicates that the output type is `sendComType`. Each element of this type contains data which should be sent to the PLC.
- Buffer Ξ dataTable. The data present in the buffer can be used, e.g., in pre-conditions.
- Pre-conditions
  - `recCom?.tn` $\in$ `tnSet\{valA}` expressing that the first component of the input must belong to `tnSet` but is not equal to `valA` (G_1),
  - $\neg$ `(pre pred)` expressing that predicate `pred` must not hold for this schema to be effective (G_2).
- Operation `sendCom! = expr` which assigns the value resulting from the evaluation of `expr` to the output variable (U).

For each of the elements mentioned in this schema, except the input and output variables, separate Z schemas are defined as well.

In the Z schema from Fig. 6, two pre-conditions need to be checked and one operation is executed. These pre-conditions can serve as a guard and the operation as an update on a transition in the automaton which can be associated with sending commands to the PLC of the surge barrier. For example, the command `c_increase` from Fig. 3 to start increasing the water level in the dock would be represented by the transition with the guard $G_1 \wedge G_2$, the update U, and event label `c_increase`. This transition can be part of a separate component model, see Fig. 6, such that it synchronizes with all other component models.

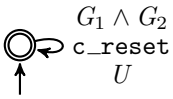

Fig. 7. The component model representing the Z schema from Figure 6.

As said, a difference between modeling with automata and with Z schemas is that automata are more specific. For each control command, a transition in an automaton is used to model it. This means that, when modeling certain behavior, Z schemas are more compact. This does come with the disadvantage that Z schemas are not executable: for describing the sending of several control commands, Z uses the same schema, while automata use a transition to model each command.

6. CONCLUSION

This paper revisited a subset of the formal Z specifications used in the design of the BOS control system operating the Maeslantkering storm surge barrier. Here we employed automaton modeling and supervisory control synthesis instead. Compared to previous case studies using supervisory control synthesis for infrastructural systems, there are several key differences with the design of the BOS for the Maeslantkering. First, the BOS supervises software systems and PLC controllers, while in those previous cases the synthesized supervisor directly interacts with actuators and sensors. Second, the BOS is rarely actively operating the Maeslantkering to close and to open (even for testing purposes), which further amplifies the need for first-time right controller software to meet the high reliability criteria of the system in total.

Supervisory control synthesis has been successfully applied to design a supervisor for part of the BOS script that contains the operational logic of the storm surge barrier. Automata and event condition models have been constructed for the components and the requirements. From these, a concise supervisory controller can be synthesized and controller software can be generated automatically. This controller is guaranteed to meet all safety requirements.

We discussed several challenges related to this modeling step. Furthermore, the same modeling language has been used to rewrite Z schemas that were used to formalize the functional view of the BOS. With an example, we showed how a Z schema could be modeled as an extended finite automaton. This work may play a pivotal role in the intended redesign of the BOS, to ensure safe operation of the Maeslantkering for decades to come.

The original specification captured the intended implemented solution, the actual underlying requirements were not (formally) captured in the documentation. These requirements are vital for supervisory control synthesis. As the possibility of testing on the actual system under ex-

treme weather conditions is effectively non-existent, the usage of digital twins may provide value in facilitating discussion with system experts. Recent work on developing digital twins for tunnel supervisory controllers, see Moormann et al. (2022), could also be applicable in the redesign of the BOS.

ACKNOWLEDGMENT

The authors thank Han Vogel from Rijkswaterstaat for his valuable feedback and support and Piotr Klimczak from Rijkswaterstaat for numerous in-depth discussions about the BOS and the Maeslantkering. Piotr Klimczak made us aware of the analogy between our way of documenting and the Rosetta stone.

REFERENCES

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, 2nd edition.

ESCET (2022). Eclipse supervisory control engineering toolset. URL `http://projects.eclipse.org/projects/technology.escet`. Last accessed 29 March 2022.

Goorden, M.A., Moormann, L., Reijnen, F.F.H., Verbakel, J.J., van Beek, D.A., Hofkamp, A.T., van de Mortel-Fronczak, J.M., Reniers, M.A., Fokkink, W.J., Rooda, J.E., and Etman, L.F.P. (2020). The road ahead for supervisor synthesis. In *Dependable Software Engineering. Theories, Tools, and Applications*, volume 12153 of *Lecture Notes in Computer Science*, 1–16. Springer. doi:10.1007/978-3-030-62822-2_1.

Holzmann, G.J. (1991). *Design and Validation of Computer Protocols*. Prentice Hall.

Holzmann, G.J. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5), 279–295.

IEC (2010). Functional safety of electrical/electronic/programmable electronic safety-related systems - parts 1 to 7. Standard IEC 61508:2010, International Electrotechnical Commission. URL `https://webstore.iec.ch/publication/22273`.

Jia, X. (1995). ZTC: A type checker for Z user's guide 2.0. Technical report, DePaul University, Chicago, USA.

Kars, P. (1996). The application of Promela and Spin in the BOS project. In *The Spin Verification System, Proceedings of a DIMACS Workshop*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 51–63. DIMACS/AMS. doi:10.1090/dimacs/032/05.

Kars, P. (1998). Formal methods in the design of a storm surge barrier control system. In G. Rozenberg and F.W. Vaandrager (eds.), *Lectures on Embedded Systems. EEF School 1996*, volume 1494 of *Lecture Notes in Computer Science*. Springer. doi:10.1007/3-540-65193-4_28.

Markovski, J., Jacobs, K.G.M., van Beek, D.A., Somers, L.J.A.M., and Rooda, J.E. (2010). Coordination of resources using generalized state-based requirements. *IFAC Proceedings Volumes*, 43(12), 287–292. doi:10.3182/20100830-3-DE-4013.00048.

Martin, J.M.R. and Welch, P.H. (1997). A design strategy for deadlock-free concurrent systems. *Transputer Communications*, 3(4), 215–232.

Miremadi, S., Åkesson, K., and Lennartson, B. (2008). Extraction and representation of a supervisor using guards in extended finite automata. In *9th International Workshop on Discrete Event Systems*, 193–199. doi:10.1109/WODES.2008.4605944.

Miremadi, S., Lennartson, B., and Åkesson, K. (2011). A BDD-based approach for modeling plant and supervisor by extended finite automata. *Transactions on Control Systems Technology*, 20(6), 1421–1435.

Moormann, L., Maessen, P., Goorden, M.A., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2020). Design of a tunnel supervisory controller using synthesis-based engineering. In *ITA-AITES World Tunnel Congress*, 573–578.

Moormann, L., van Hegelsom, J., Maessen, P., van de Mortel-Fronczak, J.M., Fokkink, W.J., and Rooda, J.E. (2022). Advantages of using digital twins in the validation of road tunnel supervisory controllers. In *ITA-AITES World Tunnel Congress*.

Ouedraogo, L., Kumar, R., Malik, R., and Åkesson, K. (2011). Nonblocking and safe control of discrete-event systems modeled as extended finite automata. *IEEE Transactions on Automation Science and Engineering*, 8(3), 560–569. doi:10.1109/TASE.2011.2124457.

Ramadge, P.J.G. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1), 206–230. doi:10.1137/0325013.

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98.

Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2017). Supervisory control synthesis for a waterway lock. In *IEEE Conference on Control Technology and Applications*, 1562–1568. doi:10.1109/CCTA.2017.8062679.

Reijnen, F.F.H., Goorden, M.A., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2020). Modeling for supervisor synthesis – a lock-bridge combination case study. *Discrete Event Dynamic Systems*, 30(3), 499–532. doi:10.1007/s10626-020-00314-0.

Sköldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *46th IEEE Conference on Decision and Control*, 3387–3392. doi:10.1109/CDC.2007.4434894.

Spivey, J.M. (1992). *The Z notation: A reference manual*. Prentice Hall.

Tretmans, J., Wijbrands, K., and Chaudron, M. (2001). Software engineering with formal methods: The development of a storm surge barrier control system; revisiting seven myths of formal methods. *Formal Methods in System Design*, 19, 195–215. doi:10.1023/A:1011236117591.

van Beek, D.A., Fokkink, W.J., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J.M., and Reniers, M.A. (2014). CIF 3: Model-based engineering of supervisory controllers. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, 575–580. Springer. doi:10.1007/978-3-642-54862-8_48.

Wonham, W.M. and Cai, K. (2018). *Supervisory Control of Discrete-Event Systems*. Springer.