Hardware-in-the-loop set-up for supervisory controllers with an application the prinses Marijke complex

Citation for published version (APA):

Reijnen, F., Verbakel, J. J., van de Mortel - Fronczak, A., & Rooda, K. (2019). Hardware-in-the-loop set-up for supervisory controllers with an application the prinses Marijke complex: the prinses Marijke complex. In *CCTA 2019 - 3rd IEEE Conference on Control Technology and Applications* (pp. 843-850). [8920561] Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/CCTA.2019.8920561>

DOI:

[10.1109/CCTA.2019.8920561](https://doi.org/10.1109/CCTA.2019.8920561)

Document status and date:

Published: 01/08/2019

Document Version:

Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Hardware-in-the-loop Set-up for Supervisory Controllers with an Application: the Prinses Marijke Complex

F.F.H. Reijnen¹, J.J. Verbakel¹, J.M. van de Mortel-Fronczak¹, and J.E. Rooda¹

Abstract—The development of supervisory controllers for cyber-physical systems is a difficult and error-prone process. Supervisor synthesis enables control designers to automatically synthesize a correct-by-construction supervisor from a model of the plant in combination with a model of the control requirements. To validate the behavior of the synthesized supervisor, model simulation is commonly used. While model simulation is a powerful tool for validation, it offers only a partial analysis. Aspects related to the execution of the supervisor on the hardware and communication with subsystems, such as human machine interfaces, cannot be validated with model simulation. To bridge the gap between model simulation and realization, hardware-in-the-loop (HIL) simulation can be performed after model simulation and before implementation. For HIL simulation, the controller realization and its subsystems are connected to a model of the plant.

The purpose of this paper is to propose an engineering method that combines synthesis-based engineering with HIL simulation. Models created for synthesis are refined and re-used to obtain models for model simulation and HIL simulation. To illustrate this method, a case study is presented that demonstrates its application to the Prinses Marijke complex, consisting of two waterway locks and a storm surge barrier. The necessary models have been developed and implemented in an HIL set-up.

I. INTRODUCTION

Due to the large number of sensors and actuators present in modern cyber-physical systems and the growing functionality that has to be provided by the control system, the development of supervisory controllers (or supervisors) for these systems is a difficult and error-prone process. Supervisor synthesis, introduced in [1], enables control designers to automatically derive a supervisor from a model of the uncontrolled system (the plant) in combination with a model of the control requirements. Subsequently, this supervisor can be implemented as a controller. The use of supervisor synthesis significantly reduces the development time and increases the quality of supervisory controllers, as the supervisor by construction adheres to the requirements and is nonblocking. This way of designing a controller for a system is referred to as synthesis-based engineering [2].

Although synthesis guarantees that the supervisor adheres to the requirements, it is not always clear if the requirements are complete and correct. Therefore, the behavior of the synthesized supervisor has to be validated. Model simulation is

often used for this purpose. It is a quick and intuitive method for validation of the controlled system behavior, before it is built. For simulation, the model of the supervisor is combined with a (hybrid) model of the plant. Model simulation allows an engineer to analyze the controlled behavior of system under different scenarios. When undesired behavior is observed, the plant or the control requirements can be adapted, and a new supervisor can be synthesized. While simulation is a valuable tool for early validation, it provides only a partial analysis. One of the main shortcomings is that a model of the supervisor is simulated, and not the actual implementation code. Another shortcoming is that the simulation platform and the implementation platform, often a programmable logic controller (PLC), have different operating semantics. Also, performance metrics such as cycle times cannot be derived from the simulation. Finally, communication to other subsystems such as an interface for operators cannot be tested. To overcome all these shortcomings, hardware-in-the-loop (HIL) simulation can be performed after model simulation and before implementation in the real system.

HIL simulation is a method in which the supervisor is implemented on the hardware, but instead of the hardware being connected to the real plant, it is connected to a model of the plant. HIL simulation offers the same advantages as model simulation, such as early validation and the possibility to simulate scenarios that would be unsafe to test on the real system. In addition, as the implementation code is run on the hardware, the performance metrics can be measured and the interaction with other subsystems can be evaluated. The increase in test coverage contributes towards improving the controlled system in the early design phase. Because of this, HIL simulation has been proven to be useful for a range of applications, such as wind turbines [3], mineral grinding [4], and traffic control [5].

HIL simulation in combination with supervisor synthesis has been explored before in [6] and [7]. In [6], a supervisor for a patient support system for an MRI has been developed. The supervisor was first validated using model simulation. Subsequently, the model of the supervisor was connected to the real hardware. Different from the HIL simulation described above, a model of the supervisor was connected to the plant realization, instead of connecting the supervisor realization to a model of the plant. This is a different type of HIL simulation. While this method can be beneficial for rapid prototyping, it requires the plant to be available. In [7], an implementation environment for automated manufacturing systems, which uses HIL concepts, is presented. Here, parts of the plant are simulated, while the supervisor is imple-

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

¹Ferdie Reijnen (f.f.h.reijnen@tue.nl), Jeroen Verbakel, Joanna v.d. Mortel-Fronczak, and Jacobus Rooda are with the Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, the Netherlands.

mented on the real PLC. In that study, a separate simulation environment has been coded in a SCADA system to represent all the components. The simulated components can gradually be replaced by the real components. In [7], manual coding of both the supervisor and the simulation environment has been applied, which is time consuming and error prone.

The main contribution of this paper is twofold. Firstly, it presents a method that combines synthesis-based engineering and HIL simulation. For this method, the models already available for supervisor synthesis are refined and re-used to automatically generate the plant model for simulation, the plant model for HIL simulation, and the PLC implementation code. Compared to [7], this approach decreases the development time and reduces coding errors. The second contribution of this paper is the description of an application of the proposed method to an industrial case, the Prinses Marijke complex. For this case, the necessary models have been developed and implemented in an experimental HIL set-up. The models and additional information are available in [8]. The project reported on in this paper has been performed in cooperation with Rijkswaterstaat (RWS), part of the Ministry of Infrastructure and Water Management of the Government of the Netherlands.

This paper is structured as follows. The Prinses Marijke complex used as the case study is introduced in Section II. In Section III, synthesis-based engineering together with HIL simulation is discussed. The set-up used for HIL simulation is presented in Section IV. In Section V, the proposed method is used to develop a supervisor and a simulation environment for the Prinses Marijke complex. Finally, Section VI concludes this paper.

II. PRINSES MARIJKE COMPLEX

As a case study, the Prinses Marijke complex is considered. The complex, depicted in Fig. 1, consists of two waterway locks (left-hand side) and a storm surge barrier (right-hand side). Under normal conditions, the barrier is open and vessels can pass under it. In case of high water levels, the barrier is closed to regulate the water flow in the Amsterdam-Rhine Canal. In this case, vessels have to travel via the locks.

Fig. 1. Overview of the Prinses Marijke complex.

A waterway lock is used in canals and rivers to raise or lower vessels between different water heights. Moreover, similar to the storm surge barrier, it is used to block and regulate the water flow. To do this, a lock consists of a large chamber (here: 300 m $\times$ 18 m), that is separated from the rest of the canal by watertight gates at both sides. The water level inside the chamber can be varied by opening or closing paddles in the gates. Vessels entering and leaving the lock are signaled by lock traffic lights. An operator is responsible for controlling the lock, i.e., opening and closing the gates and paddles, and switching the traffic light aspects.

Building of the Prinses Marijke locks was completed in 1939, while the storm surge barrier was completed in 1981. Many of the technical installations are nearing the end of their life, and a complete renovation of the complex is planned. Part of this renovation includes the supervisory control system. Currently, the controller is implemented using a relay control box. Operators interact with the controller via a control panel consisting of push buttons and feedback lamps. To increase safety and functionality, the relay circuits will be replaced by a PLC and the control panel by a graphical human machine interface (HMI).

III. SYNTHESIS-BASED ENGINEERING

Supervisor synthesis [1] is a method to automatically derive a minimally restrictive supervisor from a model of the plant (representing what the system *can* do) and a model of the control requirements (representing what the system *should* do). The supervisor restricts the behavior of the plant, such that it by construction adheres to the requirements and is nonblocking and controllable with respect to the plant.

Fig. 2 schematically shows the engineering process, adapted from [2], of designing and implementing a supervisory controller for a system. First, a set of requirements for the overall system S_R is defined, from which a global system design S_D can be made. The system design is decomposed into two parts: the plant and the supervisory controller. For the supervisor, a set of control requirements is defined and modeled, this yields C_R . For the plant, a set of requirements P_R is defined, and from this a plant design P_D , and a discrete-event plant model P_{disc} is made. From C_R and P_{disc} , a supervisory controller C_S can be automatically synthesized (see Section III-A). To validate the design, the supervisor is simulated together with a hybrid plant model P_{hyb} (see Section III-B). When the design is consistent with the intended behavior, the PLC control code C can be generated automatically (see Section III-C), and be implemented on a PLC. To validate the behavior of the controller on the PLC, and its interaction with all subsystems, HIL simulation is used (see Section III-D). For this, P_{HIL} is automatically generated from P_{hyb} . Finally, the plant is built and the PLC can be connected to the real plant.

A. Supervisor synthesis

For supervisor synthesis, the plant is modeled as a discrete-event system [9], for instance, by using extended finite-state automata (EFAs) [10]. When modeling the plant,

Fig. 2. A schematic overview of the engineering method, adapted from [2].

it is important to choose the right abstraction level. For later control implementation, the models should represent the the inputs and outputs of the control unit, as described in [11]. Events defined in the model should correspond to the switching of input or output values, i.e., a sensor switching from on to off. This results in many small component models $P = \{P_1, \dots, P_n\}$, each representing a sensor or an actuator in the system. The plant model P_{disc} is equal to the synchronous product (see, e.g., [10]) of all component models, $P_{disc} = P_1 \parallel \dots \parallel P_n$.

As an example, consider a Boolean sensor and a Boolean actuator, that can either be off or on. The sensor can be modeled as the left-hand side automaton in Fig. 3. The locations *Off* and *On* are depicted by circles. Transitions between states are depicted as arrows, where dashed arrows denote uncontrollable events, i.e., events that cannot be disabled by the supervisor. The unconnected incoming arrow denotes the initial state and a filled circle indicates a ‘rest’ or ‘final’ state. The actuator model is the right-hand side automaton in Fig. 3. The state transitions of this automaton are controllable, implying that the supervisor can disable the occurrence of these events.

Fig. 3. Component model of sensor S (left) and actuator A (right).

Similar to the plant model, the requirements can also be modeled as a collection of many small requirements $R = \{R_1, \dots, R_m\}$. Aside from using extended finite-state automata, requirements can also be modeled using event-condition requirements, introduced in [12]. Here, an event is only allowed to occur whenever the condition evaluates to *true*. Event-condition requirements allow the modeler to concisely specify the desired system behavior. The requirement model C_R is equal to the collection of R and the event-condition requirements.

Supervisor synthesis is used to automatically derive a model of the supervisory controller C_S , from P_{disc} and C_R . The result of the synthesis algorithm is for every controllable event in the system additional restrictions S , to satisfy nonblockingness and controllability, as described in [13]. The supervisor is then represented by $C_S = P_{disc} \parallel C_R \parallel S$.

B. Model simulation

It is desirable to validate the plant design and the synthesized supervisor before the system is built. Model simulation is widely used for this purpose, as discussed, e.g., in [14] and [15]. For model simulation, the model of the supervisor C_S is coupled to a model of the plant. Interactions are modeled as shared events. The model of the plant can be a discrete-event model. However, often a hybrid model [16] is preferred since it can also capture the continuous-time behavior of the plant. This makes the simulated behavior match the behavior observed in reality more closely and, in turn, it makes validation more intuitive. To this end, P_{disc} can be refined, by including time-dependent behavior, to obtain a hybrid plant model P_{hyb} . Because a discrete-event plant model is already defined for synthesis, it can be refined, such that little effort is required to obtain a hybrid plant model for simulation.

As an example, consider the two component models from Fig. 3. The model can be refined by adding continuous time behavior, via variable x . The value of x ranges between 0.0 and 3.0 and increases or decreases when the actuator is in the state *On* or *Off*, respectively, denoted by the differential equation on the right-hand side of Fig. 4. The sensor switches on when $x \geq 3.0$ and switches off when $x < 2.5$, denoted by the keyword **when**.

Additionally, the simulation can be enriched by adding interactive visualization via an SVG image, as demonstrated in [14]. An SVG is an XML-based vector image format designed for interactivity and animation [17]. The properties of the objects in this image, e.g., color, visibility, rotation, and dimensions, can depend on the locations of automata and the values of variables in the model.

Fig. 4. Hybrid sensor model refinement S_{hyb} .

C. Controller generation

When the behavior of the controlled system model is consistent with the intended behavior, the supervisor can be implemented. Because of the formal automata description of the supervisor, automatic controller code generation of, e.g., Structured Text for PLCs or Java for PCs is possible. For code generation from synthesized supervisors, dedicated tools exist, such as CIF [14] and Supremica [15].

To connect the events in the model to the inputs and outputs of the control hardware, a hardware mapping is supplied. The hardware mapping H_{map} is also supplied as an EFA. For connecting the rising and falling edge of an input signal I to an uncontrollable event, the left-hand side automaton of Fig. 5 is used. For connecting the rising and falling edge of an output signal Q to a controllable event, the right-hand side automaton of Fig. 5 is used. The keyword **do** denotes that when this event occurs, the corresponding update is executed.

Fig. 5. Hardware mapping of a sensor (left) and an actuator (right).

The EFA model for which code is generated is then $P_{disc} \parallel C_R \parallel S \parallel H_{map}$. To represent the behavior of the EFA network in a sequential programming language, the method from [18] is used. This method linearizes the behavior of the EFA system such that synchronization between events is removed. The result is one EFA with one location and for each event a selfloop with a guard and an update. The original EFAs and the resulting EFA have the same behavior. For the linearized system, implementation in sequential code is possible. It is important to notice that this method does not rely on computing the synchronous product, and in turn is not subjected to state-space explosion, which is beneficial for the size of the generated code.

D. Hardware-in-the-loop simulation

As discussed before, model simulation is a valuable tool for early validation, yet it is only a partial analysis. HIL simulation offers a method to validate the behavior of the realization of the supervisory controller (in contrast to a model of the supervisory controller) with its subsystems, e.g., an HMI or a logging system, together with a model of the plant. To implement this method, a (Java) program P_{HIL} , that

is used as the plant model in HIL simulation, is automatically generated from P_{hyb} , such that no additional modeling or coding effort is required. For generating P_{HIL} , again the method from [18] is used. The control implementation C can be connected to P_{HIL} , to perform HIL simulation. Additional, external subsystems, such as an HMI can be connected to the controller, to validate its functionality. An operator can now control the HIL simulation via the HMI, as if he were operating the real system.

IV. HARDWARE-IN-THE-LOOP SET-UP

Here, the set-up that has been developed for HIL simulation of supervisory controllers is described. A schematic representation of the set-up is depicted in Fig. 6. The set-up consists of three parts: the HMI for the operator implemented on a PC, the controller implemented on a PLC, and a model of the system (P_{HIL}) implemented on another PC. The HMI communicates with the PLC by sending commands given by the operator. The PLC communicates status information to the HMI to inform the operator about the system's state. The PLC can influence the behavior of the plant by switching actuators on or off. The plant reacts by switching sensors on or off. Physically, the communication between parts is realized using the OPC UA protocol over Ethernet.

Fig. 6. Schematic representation of the HIL set-up.

The HMI is implemented on a supervisory control and data acquisition (SCADA) system, which is the industrial standard for HMIs. Designing the HMI graphics and connecting the HMI to the controller is still a manual task. All the connections to the PLC should already be included in the discrete-event plant model for synthesis and in the hardware mapping, such that all required information is available and the controller knows which commands are possible.

The controller is implemented on a Siemens PLC, and its code in the structured control language (SCL), is automatically generated from the synthesized supervisor, as described in Section III-C. For the PLC, it does not matter whether it is communicating with the system model or with the real system.

Communicating with the real system is realized via physical inputs X and physical outputs Y of the PLC that are connected via wires to, respectively, sensors and actuators in the plant P . Additionally, a PLC has a set of internal

variables M , that is not connected to the physical inputs or outputs. These variables are primarily used as memory, but also for communication to and from the HMI. The PLC cycle works as follows. At the start, the values of X are written to input image I . The PLC program is run, which can update the values of M and output image Q . At the end of the cycle, the values of Q are written to Y and the cycle restarts. This is schematically depicted in Fig. 7. If the inputs and output hardware is not physically connected, the images I and Q can still be accessed. However, the read and write operations from X and to Y , respectively, are not performed.

Fig. 7. A schematic overview of the PLC in combination with the real plant.

Communicating with the HIL simulation is realized via virtual inputs and outputs. The system model P_{HIL} is also implemented on a SCADA system. Here, we used the Ignition SCADA software [19] to execute code. The SCADA application is able to access and manipulate values of variables in I and Q . The code operates similarly to the PLC code. It cyclically reads all actuator signals from Q , calculates all state transitions, and then writes all the sensor signals to I , as schematically depicted in Fig. 7. Hence, instead of the inputs being provided by X and outputs being supplied to Y , they are provided by and supplied to the HIL model.

Fig. 8. A schematic overview of the PLC in combination with the HIL set-up.

A hardware mapping is needed for generating code of the system model. As the system model is a refinement of the plant model used for the synthesis, it uses the same events as the controller. Hence, the hardware mapping is exactly inverse (denoted by H_{map}^{-1}), and can therefore be automatically

generated from the hardware mapping of the controller. To do so, the guards for sensors are substituted by assignments, and the assignments for actuators are substituted by guards. For example, the inverse of the mapping for events u_{on} and c_{on} in Fig. 5 is u_{on} **do** $I := T$, and c_{on} **when** $Q = T$, respectively.

The process of generating code for the HIL set-up is depicted in Fig. 9. As can be seen, during the SCL code generation, addresses of the variables in I , Q , and M are created (from H_{map}) that are used by P_{HIL} and the HMI. Because of this, there is no manual coding required to implement P_{HIL} . A step-by-step guide describing how to set-up the simulation is available in [8].

Fig. 9. A schematic overview of code generation for the HIL set-up.

V. APPLICATION: PRINSES MARIJKE COMPLEX

This section describes how the engineering method of Section III is used to develop a supervisory controller for the Prinses Marijke complex and how it is implemented in the HIL set-up of Section IV.

A. Modeling the plant and the requirements

To synthesize a supervisor, first the plant and the requirements are modeled. For the plant, many components can be represented by automaton of the same type, i.e., automaton that have the same location and transition structure. This shows the usefulness of using templates, i.e., standard models that can be instantiated per component, as first introduced in [20]. The use of templates is beneficial as it accelerates the modeling process, and also reduces the risk of modeling errors. In addition, many locks are similar, and consist of the same components, such that standard templates can be created for all locks. For this project, the templates that were developed for a different lock in Tilburg [21], have been re-used. Similarly, for the storm surge barriers, the components have been modeled by the templates already defined for locks. The modeled components are: gates, paddles, entering lock traffic lights, leaving lock traffic lights, water-height sensors, and the HMI. These components are further decomposed into standardized sensor and actuator models.

To illustrate the process of developing a component model for a lock, consider the entering lock traffic lights depicted in Fig. 10. A low-level controller only allows four sign aspects to be shown, and only allows certain state transitions. The actuator for this traffic light is modeled as the automaton in Fig. 10, where each aspect is represented by a state and switching between aspects is represented by transitions. The initial state is the red aspect and the marked states are the red and double red aspect.

TL.c_g **needs** EGate.S.Open $\wedge$ WGate.S.Open
TL.c_g **needs** Command.GreenAspect

B. Supervisor synthesis

In Table I, the number of component models $\#c$ in each subsystem are given, as well as the number of requirements $\#r$. It should be noted that the models for the north lock and the south lock are identical. The state-space sizes of the uncontrolled plant, as well as of the plant under control of

System	# <i>c</i>	# <i>r</i>	Uncontrolled	Controlled
Waterway lock north	76	144	3.7×10^{19}	1.9×10^{11}
Waterway lock south	76	144	3.7×10^{19}	1.9×10^{11}
Storm surge barrier	24	24	5.9×10^5	6.6×10^4
Prinses Marijke complex	176	312	8.1×10^{44}	2.4×10^{27}

The states of the automata during simulation are visualized via the schematic image given in Fig. 12. Elements, such as color and position, change depending on the current location of each automaton and the values of variables. Commands given via the HMI can be simulated by clicking on interactive buttons in the image. Some control windows are hidden and can be opened by clicking on a component, e.g., clicking on a traffic light opens the traffic light control window.

To generate a controller, a hardware mapping has been defined. In total, 82 inputs, 87 outputs, and 55 HMI commands

Fig. 13. HMI for the Prinses Marijke complex.

are described in the hardware mapping. From the models, a PLC program has been generated that consists of 5500 lines of SCL (available in [8]). For the generation, we used CIF. The controller has been implemented on an industrial Simatic S7-315F-2 PN/DP fail-safe PLC of Siemens.

It is known that several difficulties exist when generating a controller from a supervisor model and implementing it on a PLC, as discussed in [24] and [25]. Some problems can arise due to the difference between a supervisor and a controller, these are choice, infinite response, and causality. Some other problems can arise due to the behavior of a PLC, these are simultaneity and inexact synchronization. As of now, in the literature, no algorithms exist that can be used to verify if a controller modeled as an EFA is subject to any of these phenomena. During model simulation and HIL simulation no anomalies were observed due to any of these phenomena. However, part of further research includes proving that the controller is indeed not subject to any of these phenomena.

E. Hardware-in-the-loop simulation

For HIL simulation, the set-up described in Section IV is used. The generated control code is implemented in the PLC. The HMI, shown in Fig. 13, is designed according to the standardized format described by RWS, and connected to the PLC. The states of the traffic lights, gates, water level and the barrier are obtained from the PLC memory M .

The Java program that was generated from the simulation model contains 10,300 lines of code (available in [8]). For the generation, we used CIF. To run the Java program and to connect the program to the PLC, we used the Ignition SCADA software. Connecting the program to the PLC interfaces was straightforward as the PLC input and output addresses were also generated. No additional coding was required to implement the program.

To validate the behavior of the controlled system, several

use cases were tested. Some minor errors were found that relate to the communication between the HMI and the controller. These errors had to do with commands that could be given to the controller when they should have been forbidden by the SCADA application. No errors were found that originated from the controller, as the controller was already extensively validated with model simulation before.

The HIL set-up is an effective tool for validation of the PLC controller. Moreover, the HIL set-up can also function as a tool to demonstrate the whole development process of modeling, synthesis, and implementation. Especially the possibility to connect the controller to the HMI designed by RWS was valuable. In this way, the controller could be validated by RWS in the same manner (e.g., with factory acceptance tests) as they would validate controllers that were coded by hand. Also here, it was concluded by RWS that the behavior of the controller is as intended.

VI. CONCLUDING REMARKS

In this paper, a method that integrates HIL simulation into the synthesis-based engineering framework is presented. HIL simulation offers the possibility to validate the behavior of the controller implemented on the hardware, e.g., a PLC, before the system is built. Compared to model simulation, it executes the control code, instead of a model of the controller. Moreover, it is also possible to connect the controller to external subsystems such as an HMI. For supervisor synthesis, a plant model is already available. Hence, the additional modeling effort required to obtain a simulation model and perform model simulation is small. A model suitable for HIL simulation can automatically be generated from this simulation model.

A case study on the Prinses Marijke complex was described that successfully uses this method to obtain the necessary models for supervisor synthesis, control code

generation, and HIL simulation. For this project, the HIL set-up was also successfully used as a demonstration set-up.

ACKNOWLEDGMENTS

We thank Gerrit Bruggink and Yorrick Vissers from RWS for their help on designing the hardware components for the HIL set-up and for providing valuable feedback to the plant model and HMI design. In addition, we thank Maria Angenent and Han Vogel from RWS for their involvement in this project.

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] 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] W. Li, G. Joós, and J. Bélanger, "Real-time simulation of a wind turbine generator coupled with a battery supercapacitor energy storage system," *IEEE Transaction on Industrial Electronics*, vol. 57, no. 4, pp. 1137–1145, 2010.
- [4] W. Dai, P. Zhou, D. Zhao, S. Lu, and T. Chai, "Hardware-in-the-loop simulation platform for supervisory control of mineral grinding process," *Powder Technology*, vol. 288, pp. 422–434, 2016.
- [5] D. Bullock, B. Johnson, R. B. Wells, M. Kyte, and Z. Li, "Hardware-in-the-loop simulation," *Transportation Research Part C: Emerging Technologies*, vol. 12, no. 1, pp. 73–89, 2004.
- [6] R. J. M. Theunissen, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda, "Supervisory control synthesis for a patient support system," in *Proceedings of the 10th European Control Conference*. IEEE, 2009, pp. 4647–4652.
- [7] R. A. Diogo, E. A. P. Santos, A. D. Vieira, E. d. F. R. Loures, and M. A. Busetti, "A computational control implementation environment for automated manufacturing systems," *International Journal of Production Research*, vol. 50, no. 22, pp. 6272–6287, 2012.
- [8] F. F. H. Reijnen, J. J. Verbakel, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Models and implementation code for the Prinses Marijke complex," 2019, [www.github.com/ffhreijnen/PrinsesMarijkeComplex](https://github.com/ffhreijnen/PrinsesMarijkeComplex).
- [9] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.
- [10] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *Proceedings of the 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 3387–3392.
- [11] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1040–1059, 1993.
- [12] 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 *Proceedings of the 49th IEEE Conference on Decision and Control*. IEEE, 2010, pp. 3481–3486.
- [13] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic computation of reduced guards in supervisory control," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.
- [14] 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 *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 575–580.
- [15] R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica—an efficient tool for large-scale discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5794–5799, 2017.
- [16] T. A. Henzinger, "The theory of hybrid automata," in *Verification of Digital and Hybrid Systems*. Springer, 2000, pp. 265–292.
- [17] A. Quint, "Scalable vector graphics," *IEEE MultiMedia*, vol. 10, no. 3, pp. 99–102, 2003.
- [18] 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.
- [19] Inductive Automation, "Ignition," <https://inductiveautomation.com/>.
- [20] L. Grigorov, B. E. Butler, J. E. Cury, and K. Rudie, "Conceptual design of discrete-event systems using templates," *Discrete Event Dynamic Systems*, vol. 21, no. 2, pp. 257–303, 2011.
- [21] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory control synthesis for a waterway lock," in *Proceeding of the 1st IEEE conference on Control Technology and Applications*. IEEE, 2017, pp. 1562–1563.
- [22] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based approach for modeling plant and supervisor by extended finite automata," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 6, pp. 1421–1435, 2012.
- [23] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.
- [24] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Proceedings of the 37th IEEE Conference on Decision and Control*, vol. 3. IEEE, 1998, pp. 3305–3310.
- [25] P. Malik and R. Malik, "Modular control-loop detection," in *Proceedings of the 8th International Workshop on Discrete Event Systems*. IEEE, 2006, pp. 119–124.