Design of a Tunnel Supervisory Controller using Synthesis-Based Engineering

L. Moormann[1], P. Maessen[2], M.A. Goorden[1], J.M. van de Mortel-Fronczak[1], and J.E. Rooda[1]
[1]*Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands*
[2]*Department of Grote Projecten en Onderhoud, Rijkswaterstaat, Utrecht, The Netherlands*
E-mail: l.moormann@tue.nl

ABSTRACT: Nowadays, each tunnel is equipped with a supervisory controller that ensures correct cooperation between the tunnel subsystems, such as lighting, ventilation, and emergency detection sensors. Practice has shown that traditional design methods require a lot of manual effort, which is error-prone, time consuming, and costly. Therefore, an alternative design method is explored. In this paper, three methods for designing a supervisory controller are discussed: traditional engineering, model-based engineering, and synthesis-based engineering. They are assessed based on three criteria, being the quality of the controller, the variability of the time-to-market, and the evolvability. The synthesis-based engineering method turns out to be the most appropriate design method. In a case study, a supervisory controller for a roadway tunnel in the Netherlands is designed using synthesis-based engineering and validated using simulation-based visualization. This case study shows that SBE is a suitable design method for designing a tunnel supervisory controller.

KEYWORDS: Controller Design, Supervisory Controller, Synthesis, Simulation, Validation, Verification

1. INTRODUCTION

Over the years, the method of designing infrastructural systems has changed drastically. Where traditionally everything was designed using documents, the use of computer models has now taken over a majority of that work. Since the 1970s building information modeling (BIM), a concept explained by Hardin and McCool (2015) and Kensek (2014), has been used increasingly. In BIM, multiple facets of infrastructural design are incorporated into a single model. For instance, architectural, structural, electrical, and mechanical aspects of the design are included in the same model. One benefit of this is, for instance, collision detection. This helps finding discrepancies between the incorporated aspects, as is mentioned by Azhar (2011).

One aspect that is often excluded from the design in BIM is the supervisory controller. In Figure 1, the positioning of the supervisory controller is schematically visualized. Figure 1 shows the control layers of an infrastructural system, as inspired by control structure visualizations in Lee et al. (2015) and in Melnyk (2016). Layer 1 consists of the mechanical components in the system that need to be controlled. When considering a roadway tunnel, this includes, for instance, boom barriers, traffic lights, and ventilation. Each of these components is connected to actuators and sensors, as shown in Layer 2, and its own resource controller, which constitutes the Layer 3. Layer 4 is the supervisory controller. This is the main controller that ensures correct cooperation between all these subsystems. Layer 5 is the man-machine interface. This is the interface through which the operator interacts with the system by sending commands and retrieving information.

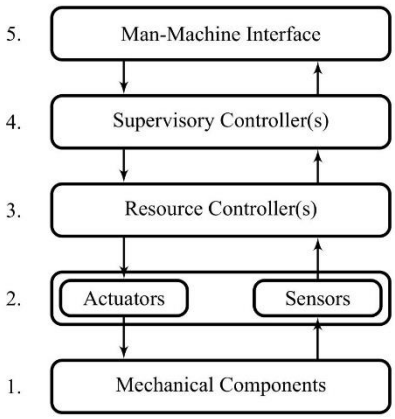

Figure 1 Schematic overview of an infrastructural system.

Due to the many interactions between the structural design of the tunnel and its supervisory controller, it is important to design these aspects concurrently. The supervisory controller design depends on infrastructural features such as tunnel dimension, geological location, and escape concept. Vice versa, the infrastructural design depends on features related to the implementation of the supervisory controller, e.g., sensor positioning and cabling.

One of the main challenges in designing a supervisory controller is proving its correctness. In traditional engineering methods, the supervisory controller is usually manually programmed based on documented controller specifications. The supervisory controller is then tested on the realized tunnel, and mistakes that are found are corrected. The correctness of the controller therefore depends on the quality of thetest, which is often limited. In more recent engineering methods, as surveyed in Estefan et al. (2007), computer models are used to perform tests in an earlier design stage using model simulations. This increases the testing possibilities of the supervisory controller and thus increases the chance that errors are found. The correctness is, however, still not proven. In the last decades, research advancements are made on the subject of automatically synthesizing supervisory controllers. This method is called synthesis-based engineering and it uses mathematical models to generate the supervisory controller through algorithms. The main advantage of using these algorithms is that the correctness of the synthesized controller is proven.

The contribution of this paper is showing that the synthesis-based engineering method is suitable for designing a supervisory controller for a roadway tunnel. Synthesis-based engineering is explained in more detail and applied to a case study. This case study covers the process of modeling of the components and the requirements, synthesizing the supervisory controller, and validating the synthesized controller.

The structure of the paper is as follows. First, Section 2 describes in detail three different engineering methods, being traditional engineering, model-based engineering, and synthesis-based engineering. In Section 3, a case study of a roadway tunnel is described where synthesis-based engineering is applied. Finally, in Section 4, the concluding remarks are presented and future work is addressed.

2. CONTROLLER DESIGN METHODS

This section discusses three methods for designing a supervisory controller, being traditional engineering, model-based engineering, and synthesis-based engineering. The methods are evaluated by looking at three criteria related to the controller:

- Quality: to what degree does the controller meet its requirements?
- Variability of the time-to-market: how large is the standard deviation of the estimated time-to-market of the controller?
- Evolvability: how easily can the controller be extended with new functionalities or adapted to similar applications?

Note that time-to-market and costs are not specifically considered here. They can be derived from the other criteria.

Firstly, when designing a controller for a new system, the time-to-market depends on the quality of the controller as errors made in the controller design negatively impact the time-to-market. This is especially the case when controller testing is done after controller realization and implementation. Secondly, when designing a controller that resembles previously designed controllers, the evolvability of the design method influences the time-to-market. A design method with a high evolvability will in this case result in a shorter time-to-market.

The costs of a controller design method mostly depend on the time-to-market, as a shorter time-to-market results in lower costs. The variability of the time-to-market also affects the costs, as costs will increase when delays occur in a controller design process.

2.1 Traditional engineering

The traditional engineering method of designing a supervisory controller is schematically shown in Figure 2. Here, each step is denoted with an arrow and each symbol represents the product of that step. A document icon indicates a documented product, and a square indicates a realized product.

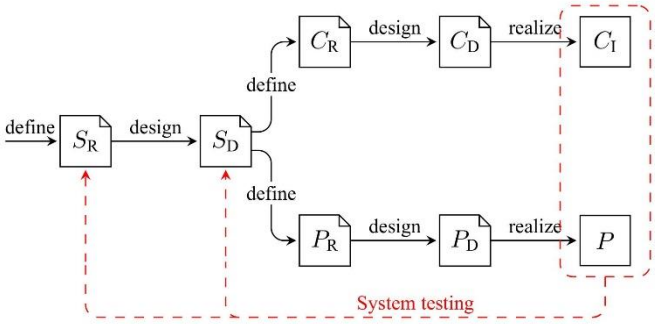

Figure 2 Schematic overview of the traditional engineering method.

First, a set of high-level requirements S_R is defined for the system, which in this case is a roadway tunnel. Based on these requirements, an initial document-based design S_D is created for the system. In the next step the system design is divided in two parts: plant P and controller C. The plant consists of the physical components in the tunnel that are relevant for the supervisory controller. These components mainly include actuators that need to be controlled, and sensors that provide information. The physical components that are not related to the supervisory controller are excluded here. For the design of both the plant and the controller, first requirements are defined (P_R and C_R), and a document-based design is created following these requirements (P_D and C_D). The final steps are building of tunnel, programming the controller, and implementing the realized controller in the tunnel.

One of the most important steps is the verification and validation of the controller, as is discussed by Wallace and Fuijii (1989). In Boehm (1979) and Frey and Litz (2000), software verification is defined as answering the question "Are we building the product right?", whereas software validation answers the question "Are we building the right product?". Verification thus checks if the specifications are correctly implemented and validation checks if the software product satisfies the intended use.

In the traditional engineering method only validation is possible. This is done through system testing, after the realization and implementation of the controller, as is indicated in Figure 2. In this step, the realized controller is tested in combination with the realized plant to validate if the controlled behavior is as intended. The correct implementation of the specifications cannot be guaranteed, unless all states and scenarios in the system are checked, which is not feasible for larger systems.

As is shown in Figure 2, the system testing reflects on the initial system requirements and design. They are performed after the realization of the controller and the plant. The main disadvantage of this is that when an error is found during this step, the designer needs to go back to the document-based design steps of the system, controller, and plant to correct this mistake. This often costs more than correcting it during the design phase, as indicated in Boehm and Basili (2007).

The controller is realized through manual programming. For large systems containing numerous components, such as a road tunnel, this is a cumbersome and error-prone task. Hence, the traditional engineering method has been assessed as follows:

- Quality: Moderate for small systems. Low for large systems, as a large number of components results in a complex cooperation between these components, and as extensive testing is not well feasible.
- Variability: Large, as errors are found at the end of the design process.
- Evolvability: Low, due to manually written code.

2.2 Model-based engineering

In the last decades, it has become more common to use executable models when designing systems. A design method using such executable models is described in Braspenning et al. (2006). Furthermore, executable models enable testing of realized components with yet to realize, virtual, components, called hardware-in-the-loop (HIL) testing, as detailed in Bullock et al. (2004). Figure 3 shows the model-based engineering (MBE) method. The method is similar to the traditional engineering method with two extra steps, denoted by arrows, and two extra products, denoted with circles to indicate models.

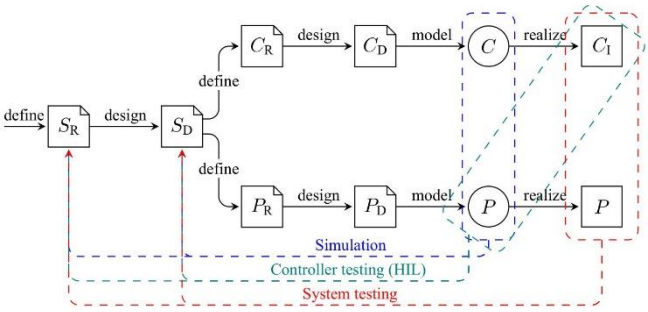

Figure 3 Schematic overview of the model-based engineering method.

As shown in Figure 3, for both the controller and the plant a modeling step is added after documenting the design. Plant model P is a model of the possible system behavior based on documented plant design P_D. Controller model C is a model of the controller based on documented controller design C_D.

Figure 3 also shows the validation steps that are possible in MBE. The first validation step shown is through simulation, as explained in Wallace and Fuijii (1989) and Pace (2004). In this step, the plant model is simulated along with the controller model to validate if the controlled behavior is the same as the desired behavior. Verification is not possible using simulations for the same reasons as mentioned for system testing in traditional engineering. Verification can, however, be performed using formal verification methods, such as the methods described in Baier and Katoen (2008).

After the controller model is validated, it is realized in PLC code. The next step is controller testing. Here, the realized controller is tested on the plant model. The benefit of this is that the actual controller can be tested before plant realization. The final testing step is the same as the validation in traditional engineering, which is the integrated system test with the realized controller and realized plant.

The main benefit of MBE is that most of the validation can be performed before controller realization and plant realization. This means that errors in the plant design and the controller design can be found earlier than in traditional engineering, which results in a more time- and cost-effective error reparation, as advocated in Boehm and Vasili (2007). Furthermore, Pace (2004) mentions the challenge of understanding the system behavior of systems with a large number of components. Simulations can help to tackle this challenge by giving

more insight in the system behavior. The MBE method has been assessed as follows:

- Quality: Large for small systems. Moderate for large systems, as insight can be gained through simulations.
- Variability: Moderate, since errors are found before the realization steps, resulting in more time-effective reparation.
- Evolvability: Moderate, since the created models can be adapted and reused for similar systems.

2.3 Synthesis-based engineering

The MBE method enables early validation through simulation and controller testing, but the correctness of the controller still very much depends on the test engineer, as is also mentioned in Taipale et al. (2011). In the 1980s, research was started on the control of discrete-event systems. The main results of this research are found in Ramadge and Wonham (1987) and Ramadge and Wonham (1989). The idea is to model the possible system behavior and the controller specifications using executable models. This enables the use of synthesis algorithms, described in works as Vahidi et al. (2006) and Ouedragogo et al. (2011), to automatically generate the supervisory controller. These synthesis algorithms are mathematically proven to generate a controller that adheres to the plant and controller requirement models. This means that verification is no longer necessary, since the synthesized controller is guaranteed to adhere to the specified requirements. Figure 4 shows the synthesis-based engineering (SBE) method. In SBE, the controller design step has been removed and instead a requirements model C_R is created directly from the documented controller requirements. Furthermore, the controller model is no longer manually created, but now synthesized from the controller requirements model and the plant model.

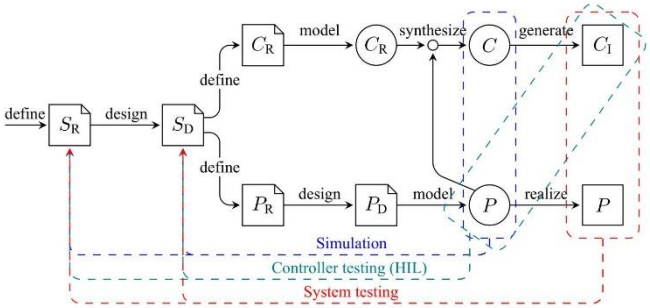

Figure 4 Schematic overview of the synthesis-based engineering method.

As can be seen in Figure 4, the same validation steps are possible as in model-based engineering. This means that the benefits described in Section 2.2 still hold for SBE. The SBE method has been assessed as follows:

- Quality: High, since the controller is proven correct.
- Variability: Low, because errors are found early in the design process, and requirements are guaranteed to be satisfied so there is no variability in verification time.
- Evolvability: High, since the models created for the plant and requirement models allow for easy reuse and adaptation.

The method used to create plant model P and requirements model C_R are now explained in Section 2.3.1 and 2.3.2, respectively.

2.3.1 Plant

The first set of models to be created is the plant. As mentioned earlier, mathematical models are required to synthesize a supervisory controller. Automata, as described by Cassandras and Lafortune (2009), are used to describe the system behavior using states and transitions. Figure 5 shows an example of an automaton definition for an actuator. The circles indicate the states of the actuator. They represent the modes the actuator can be in, i.e., the on or off mode. The arrows between the states indicate the transitions. They are transitions that can change the state of the actuator, i.e., turning it on or off. The initial state of the component is indicated by the inward arrow on the left of the off state. Marked states are indicated by concentric circles. A marked state of a component represents a rest state or a state where a task is finished. For the actuator in Figure 5, the off state is the marked state.

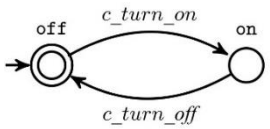

Figure 5 Example of an automaton for an actuator.

A second example of an automaton definition is shown in Figure 6. Here, the behavior of a sensor is modeled. The difference between an actuator and a sensor is the type of the transitions. When looking from the supervisory controller's perspective, it has control over turning the actuator on and off, yet the supervisory controller has no control over turning the sensor on and off. This is modeled in automata using so-called uncontrollable events. These are indicated using dashed arrows, as shown in Figure 6.

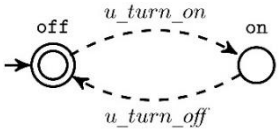

Figure 6 Example of an automaton for a sensor.

Automaton definitions such as the ones shown in Figure 5 and 6 can be instantiated for each component in the tunnel that contains this behavior, in this case for each actuator and sensor.

2.3.2 Requirements

The second set of models contains the controller requirements. These requirements are the specification, i.e., rules, that the controller must follow to ensure safe and correct controlled behavior. The requirements are modeled with state-based expressions, as introduced in Markovski et al. (2010). A requirement always specifies the desired behavior for an instantiated plant model. An example of a requirement is shown in Equation (1). In this example, `actuator_X` is an instantiation of the actuator definition and `sensor_Y` is an instantiation of the sensor definition. The requirement expresses that `actuator_X` is only allowed to turn on when `sensor_Y` is on.

```
actuator_X.c_on needs sensor_Y.on        (1)
```

3. CASE STUDY

In this section, a case study of the design of the supervisory controller for a roadway tunnel is described. The goal of this case study is to show that the synthesis-based design method is suitable to design a tunnel supervisory controller.

3.1 System description

The tunnel chosen for this case study is the Eerste Heinenoordtunnel (EHT, First Heinenoordtunnel in English), which is located south of Rotterdam, the Netherlands. Figure 7 shows two tunnels, being the EHT on the right, and the Tweede Heinenoordtunnel (THT, Second Heinenoordtunnel in English) on the left. The EHT is a two-tube roadway tunnel that was initially opened in 1969. It is maintained by Rijkswaterstaat, which is an executive body of the Dutch ministry of infrastructure and water-management. The EHT is an immersed tube tunnel in the river the Oude Maas. It was built with the main purpose to improve the connection between Rotterdam and the area south of the river. The THT was added later in 1999 and is only accessible for slow traffic such as cyclists and agricultural traffic. Only the EHT is included in this case study. The EHT is chosen for this case study because Rijkswaterstaat is currently in the preparation and planning phase of renovating this tunnel. In this renovation project both the

physical tunnel components and the tunnel supervisory controller are renewed.

In this case study, two models have been created: a basic model and the EHT model. The basic model contains all relevant components of a roadway tunnel with two traffic tubes and a middle-tunnel-channel. However, for each component that exists multiple times in a traffic

Figure 7 Birds-eye view of the Eerste Heinenoordtunnel (right) and the Tweede Heinenoordtunnel (left). Image from https://beeldbank.rws.nl, Rijkswaterstaat.

tube, e.g. the escape doors, only one is included. This means that the basic model does not represent all possible behavior of the EHT, though is does contain all the relevant requirements and interactions between the different component types. The second model, being the EHT model, is the instantiated version of the basic model containing the actual number of components in the EHT.

3.2 Modeling

In this section, some examples of the plant models and the requirement models created for the basic model[1] and the EHT model are shown. Specifically, the components used to close a traffic tube are described in detail. These components include the actuators and the sensors of the boom barrier and the traffic light.

3.2.1 Plant

First, the models of these components are created. In the plant model all possible behavior is captured. The possible behavior of the boom barrier is modeled by modeling the movement and the position of the boom barrier. The movement can be modeled using two actuators: one to move up and one to move down. The behavior of these actuators is the same, so the automata are also the same. Figure 8 shows this automaton. This automaton is instantiated twice, once for the upward actuator of the boom barrier (`actuator_up`) and once for the downward actuator of the boom barrier (`actuator_down`). The automata of the two actuators have an identical initial and marked state: when the boom barrier is idle. The initial and marked state of both automata is thus of the off state.

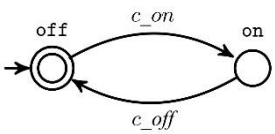

Figure 8 Automaton of a definition of an actuator.

The position of the boom barrier can be modeled using two sensors: one to detect when the boom barrier is fully closed (`sensor_closed`) and one to detect when the boom barrier is fully opened (`sensor_open`). The initial and marked state of the boom barrier is when the boom barrier is fully opened, so the two sensors have different initial and marked states.

Figure 9 Automata of the sensor detecting when the boom barrier is open (left) and the sensor detecting when the boom barrier is closed (right).

The traffic light that is used to be able to close the traffic tube has a yellow and a red aspect. The aspects of a traffic light are the lights that can turn on and off. These aspects are used in the modes of the traffic light. The possible modes of the traffic light are off, flashing yellow, full yellow, and red. Figure 10 shows the automaton for this traffic light. The transitions between the states are the possible transitions as specified by Rijkswaterstaat.

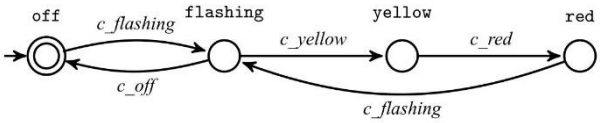

Figure 10 Automaton of the traffic light.

3.2.2 Requirements

The requirement model consists of state-based expressions that define when a component is allowed or not allowed to do something. Combining the requirement model with the plant model restricts the possible behavior to the desired behavior. In this section, several requirements are shown related to the boom barrier and traffic light models shown in the previous section. For each requirement, the textual description is given as well as the state-based expression.

1. The upward actuator of the boom barrier is only allowed to turn on if the downward actuator is turned off.

   `actuator_up.`*`c_on`* **`needs`** `actuator_down.off`

2. The upward actuator of the boom barrier is only allowed to turn on if the position of the boom barrier is not open.

   `actuator_up.`*`c_on`* **`needs`** `¬ sensor_open.on`

3. The upward actuator of the boom barrier is only allowed to turn off if the position of the boom barrier is open.

   `actuator_up.`*`c_off`* **`needs`** `sensor_open.on`

4. The downward actuator of the boom barrier is only allowed to turn on if the traffic light is showing a red light.

   `actuator_down.`*`c_on`* **`needs`** `traffic_light.red`

5. The traffic light is only to start showing a flashing yellow aspect if it not showing a red light or if the boom barrier is not closed.

   `traffic_light.`*`c_flashing`* **`needs`**
   `¬ traffic_light.red ∨ ¬ sensor_closed.on`

The set of requirements shown here is a representative set of requirements for this case study. Note that this is not the complete set of requirements[1].

3.3 Synthesis results

As described in Section 2.3, the next step in SBE is synthesizing the supervisory controller from the plant model and the requirement model. The synthesis results for the basic model and the EHT model

[1] The complete set of component models and requirement models for the basic model is available at: https://github.com/LMoormann/Basic_Model

are shown in Table 1. First, the number of component definitions, component models, and requirement models are given. Furthermore, the number of states in the uncontrolled system is shown. The synthesis algorithm that is used is multilevel synthesis, as introduced in Komenda et al. (2016). For this algorithm, the control problem is divided into smaller sub-problems. The algorithm then synthesizes a supervisor for each sub-problem, as opposed to synthesizing a single supervisor as is done in monolithic synthesis. Table 1 shows the number of supervisors that are synthesized for the basic model and for the EHT model. The advantage of using multilevel synthesis is that supervisors can be synthesized for systems with a number of states in the uncontrolled system that is too large for monolithic synthesis. Finally, Table 1 shows the sum of the number of states in each supervisor.

Table 1 Results for the basic model and the EHT model.

	Basic model	**EHT model**
Component definitions	19	19
Component models	149	540
Requirement models	355	1668
Number of states in the uncontrolled system	$1.47 \cdot 10^{48}$	$1.87 \cdot 10^{226}$
Supervisors	43	48
Sum of the number of states in each supervisor	$1.74 \cdot 10^{12}$	$1.99 \cdot 10^{56}$

When looking at the results shown in Table 1 one can see that the number of component models, the number of requirement models, and the number of states of the EHT model are considerably higher than the basic model, as can be expected. It does, however, stand out that the number of supervisors of the EHT model is not much higher compared to the basic model. Further investigation showed that often one supervisor is synthesized for components of the same type. For instance, in the basic model one supervisor controls the ventilation unit of one traffic tube, and in the EHT model all fourteen ventilation units are also controlled by one supervisor. The small increase in the number of supervisors results from a few exceptions on the previous statement, e.g., a second supervisor is synthesized for an extra water pump that is added in the EHT model compared to the basic model.

3.4 Validation

The synthesized supervisory controller is validated using simulations. In these simulations, inputs are given by the user to the supervisory controller and the response behavior of the supervisory controller is analyzed by the user. Incorrect behavior is detected and corrected in the component models or the requirement models. The supervisory controller is then synthesized using the corrected plant model and requirement model, after which the simulations can be used to check whether the corrections made are correct or whether there is more incorrect behavior.

One of the most intuitive simulation methods is simulation-based visualization, as advocated in Rohrer (2000). In this method, a visualization is created of the relevant components of a system and its control interface. In the simulation, this visualization is both used as an input interface, e.g., pressing a button, and as an output interface, e.g., showing the traffic light mode. The visualization for the basic model is shown in Figure 11. In the center of this figure, a simplified version of the basic model of the tunnel is shown containing the tunnel components used to detect an emergency and the components used to close a traffic tube. At the top and at the bottom of the figure the interfaces are shown that can be used to interact with the tunnel. For instance, these interfaces contain buttons to place an obstacle beneath the boom barrier to test the obstacle detection sensor of that boom barrier.

Simulation-based visualization has been used to validate the controlled behavior of the basic model. The EHT model does not need to be validated separately as the only difference between the EHT model and the basic model is the number of components of the same type and no new functionalities or requirements are added.

4. CONCLUDING REMARKS AND FUTURE WORK

This paper describes a method in which the supervisory controller of an infrastructural system is designed concurrently to the civil design. Three engineering methods, traditional engineering, MBE, and SBE, are evaluated by estimating the quality of the controller, the variability of the time-to-market of the controller, and the evolvability of the controller. SBE shows to be the most appropriate design method, mainly due to the proven correctness of the controller, the possibilities for testing at an early design stage, and the high evolvability of the models.

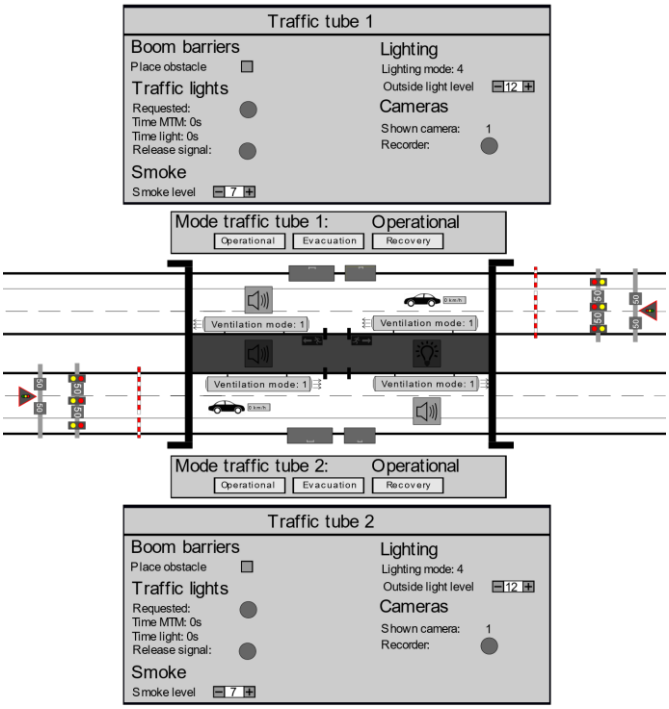

Figure 11 Visualization of the basic model used to perform simulations.

The SBE design method is applied in a case study for the design of a supervisory controller of the EHT. In this paper, the modeling steps of both the plant model and the requirement model are shown for a part of the system. The synthesis procedure is described along with numerical results, and the validation method using simulation-based visualization is explained.

This case study showed that it is possible to synthesize a supervisory controller for the EHT using SBE. As can be concluded from the synthesis results in Table 1, all component models can be efficiently modeled using a set of component definitions. These definitions also facilitate an easy extension of the basic model to the EHT model, as the same definitions can be used. The number of states in the uncontrolled system of both the basic model and the EHT model showed to be too large to synthesize a monolithic supervisor. Instead, multilevel synthesis has been used to synthesize a set of supervisors. The final step of this case study consisted of validation of the synthesized supervisors. Since simulation-based visualization has been used, the behavior of the controlled system was intuitively validated by running through various test scenarios and analyzing the controlled behavior.

Future work related to this case study includes the extension of the EHT model with manual control. Currently, only the automatic behavior of the EHT is modeled, yet many of the components in the actual tunnel should be controlled manually as well. Furthermore, the addition of fault-tolerant control is part of future work. With this addition the controlled behavior is also guaranteed to be correct when certain faults in the system are diagnosed. Finally, this case study only describes the method of synthesizing the supervisory controller and

its validation. The subsequent step is to generate PLC code from this supervisory controller, perform controller tests, and eventually system tests.

5. ACKNOWLEDGEMENTS

The authors are grateful to Rijkswaterstaat for funding this research, and to Han Vogel for his support in the MultiWaterWerk project. Furthermore, we thank Pascal Etman for the stimulating discussions and his guidance throughout this research project.

6. REFERENCES

Azhar, S. (2011) "Building information modeling (BIM): Trends, benefits, risks, and challenges for the AEC industry," Leadership and management in engineering, vol. 11, no. 3, pp241-252.

Baier, C, and Katoen, J. P. (2008) Principles of model checking. MIT press.

Boehm, B. W. (1979) "Software engineering: R&D trends and defense needs," Research directions in software technology.

Boehm, B. W., and Basili, V. R. (2007) "Software defect reduction top 10 list," Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research, vol. 34, no. 1, p75.

Braspenning, N. C. W. M., Van de Mortel-Fronczak, J. M., and Rooda, J. E. (2006) "A model-based integration and testing method to reduce system development effort," Electronic Notes in Theoretical Computer Science, vol. 164, no. 4, pp13-28.

Bullock, D., Johnson, B., Wells, R. B., Kyte, M., and Li, Z. (2004) "Hardware-in-the-loop simulation," Transportation Research Part C: Emerging Technologies, vol. 12, no. 1, pp73-89.

Cassandras, C. G., and Lafortune, S. (2009) Introduction to discrete event systems. Springer Science & Business Media.

Estefan, J. A. (2007) "Survey of model-based systems engineering (MBSE) methodologies," Incose MBSE Focus Group, vol. 25, no. 8, pp1-12.

Frey, G., and Litz, L. (2000) "Formal methods in PLC programming," in Cybernetics evolving to systems, humans, organizations, and their complex interactions, vol. 4. IEEE, pp2431-2436.

Hardin, B. and McCool, D. (2015), BIM and construction management: proven tools, methods, and workflows. John Wiley & Sons.

Kensek, K.M. (2014) Building information modeling. Routledge.

Komenda, J., Masopust, T., and Van Schuppen, J. H. (2016) "Control of an engineering-structured multilevel discrete-event system," in 13th International Workshop on Discrete Event Systems. IEEE, pp103-108.

Lee, L., Bagheri, B., and Kao, H. (2015) "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," Manufacturing letters, vol. 3, pp18-23.

Markovski, J., Van Beek, D. A., Theunissen, R. J. M., Jacobs, K. G. M., and Rooda, J. E. (2010) "A state-based framework for supervisory control synthesis and verification," in 49th IEEE Conference on Decision and Control (CDC). IEEE, pp3481-3486.

Melnyk, A. (2016) "Cyber-physical systems multilayer platform and research framework," Advances in cyber-physical systems, no. 1, pp1-6.

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, vol. 8, no. 3, pp560-569.

Pace, D. K. (2004) "Modeling and simulation verification and validation challenges," Johns Hopkins APL Technical Digest, vol. 25, no. 2, pp163-172.

Ramadge, P. J. G., and Wonham, W. M. (1987) "Supervisory control of a class of discrete event processes," SIAM journal on control and optimization, vol. 25, no. 1, pp206-230.

Ramadge, P. J. G., and Wonham, W. M. (1989) "The control of discrete event systems," Proceedings of the IEEE, vol. 77, no. 1, pp81-98.

Rohrer, M. W. (2000) "Seeing is believing: the importance of visualization in manufacturing simulation," in 32nd conference on Winter simulation. Society for Computer Simulation International, pp1211-1216.

Taipale, O., Kasurinen, J., Karhu, K., and Smolander, K. (2011) "Tradeoff between automated and manual software testing," International Journal of System Assurance Engineering and Management, vol. 2, no. 2, pp114-125.

Vahidi, A., Fabian, M., and Lennartson, B. (2006) "Efficient supervisory synthesis of large systems," Control Engineering Practice, vol. 14, no. 10, pp1157-1167.

Wallace, D. R., and Fujii, R. U. (1989) "Software verification and validation: an overview," IEEE Software, vol. 6, no. 3, pp10-17.