Systematic Model-Based Design and Implementation of Supervisors for Advanced Driver Assistance Systems

Tim Korssen, Victor Dolk, *Student Member, IEEE*, Joanna van de Mortel-Fronczak, Michel Reniers, *Senior Member, IEEE*, and Maurice Heemels, *Fellow, IEEE*

***Abstract*— The number of advanced driver assistance systems (ADASs) and the level of automation in modern vehicles is increasing at a rapid pace. Moreover, multiple of these ADASs can be active at the same time and therefore may need to interact with each other. As a consequence, the design of the supervisor layer that is responsible for proper coordination of the control tasks performed by the low-level ADASs controllers is becoming more complex and safety-critical. For this reason, there is a strong need for automated synthesis tools that lead to supervisors that are safe by design. In this paper, we present a systematic approach to model-based supervisor design using discrete-event system representations. In particular, this paper shows that the proposed method is suitable to deal with the multiple and complex systems of interacting ADASs. To be more specific, in contrast to current practice, which often relies on textual specifications and exhaustive testing, the proposed method has four main advantages: 1) it is based on mathematically specified requirements that only allow one interpretation; 2) it prevents blocking situations by design; 3) it guarantees correctness in the sense that the resulting supervisor satisfies all the specified requirements; and 4) code is generated from the obtained supervisor which eliminates the need for manual coding. The proposed method is demonstrated by means of a case study on cruise control and adaptive cruise control. The resulting supervisor is validated by simulations and experiments on a modern passenger vehicle. Based on the results presented in this paper, it can be concluded that the model-based supervisor design, simulation, and implementation method is promising and powerful for future applications in the automated vehicle systems.**

***Index Terms*— Supervisory control, model-driven development, control system synthesis, automotive engineering, advanced driver assistance systems.**

I. Introduction

TODAY'S modern vehicles contain many Advanced Driver Assistance Systems (ADASs). Examples of such ADASs are various forms of (adaptive) cruise controllers, lane-keeping assistance systems and collision-avoidance systems. Initially, the aim of these ADASs was to enhance driver's comfort, mostly for highway traffic. Today, ADASs are specifically being developed to increase traffic safety and throughput, and to reduce fuel consumption and air pollution [1]–[5]. Moreover, they should be able to perform well in a large variety of traffic situations as the level of automation is growing at a rapid pace. As such, the various assistance systems present in modern vehicles have a strong interaction and therefore, can no longer be regarded as separate systems. According to the overview provided in [6], it is expected that in the near future ADASs fully cooperate with the powertrain management. As a consequence, future ADASs will become more safety-critical and complex.

Manuscript received January 15, 2017; revised June 30, 2017 and October 7, 2017; accepted November 9, 2017. Date of publication December 19, 2017; date of current version February 1, 2018. The Associate Editor for this paper was P. Zingaretti. *(Corresponding author: Michel Reniers.)*

The authors are with the Department of Mechanical Engineering, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands (e-mail: m.a.reniers@tue.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2017.2776354

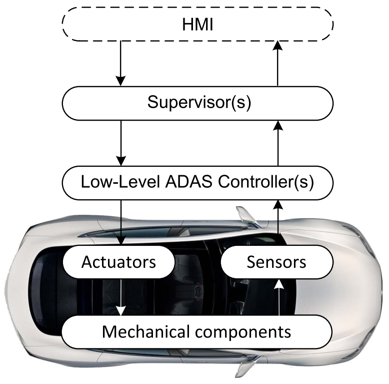

Fig. 1. Hierarchical control for highly inter-coupled complex vehicle systems.

A typical control architecture for ADASs is shown in Fig. 1, see also [7]–[9]. Observe that actuators and sensors are directly connected to the low-level ADASs controllers. These low-level ADAS controllers determine the control input that is sent to the actuators. Since ADASs typically act in a dynamic environment, multiple low-level controllers might be used to determine the control input for the same actuator. For example, in Adaptive Cruise Control (ACC), depending on the inter-vehicle distance and the relative speed, various low-level ADAS controllers are used to drive the throttle and brake. The purpose of the supervisor layer is to coordinate the low-level controllers according to the driver's decisions and sensor data. The driver's decisions are communicated to the supervisor through the Human-Machine-Interface (HMI). The goal of the supervisor thus is to enforce safe and reliable behavior of the overall system during operation.

Although designing of ADAS systems has been studied for many years, only recently formal synthesis, validation and verification methods for ADASs have been proposed in the literature. In [10], distributed hybrid system verification is used to verify that a control system for cars equipped with ACC is collision-free. In [11], an ACC scheme is proposed that combines a model-predictive control scheme with a formally verified safety controller. In [12], safety specifications for ACC systems are verified via the construction of control barrier functions. The work in [13] exploits game theoretic techniques to synthesize a hybrid controller for automated highway systems that is safe by design. In [14], a synthesis method is proposed that relies on linear temporal logic specifications and that results in correct-by-construction control software for ACC. The aforementioned works focus on the correctness of individual low-level ADAS controllers. Despite the growing complexity of ADASs, only a few references address the design of supervisory controllers. Specifically, [13], [15], [16], proposed supervisors for platooning maneuvers such as merging and splitting. The correctness of supervisor designs was established by debugging.

Proper design of a supervisor for a highly complex system is in general challenging due to the large dimension of the state space of the system [17]. For example, the platoon maneuver supervisor proposed in [15] already involves about 500,000 states and 10,000,000 transitions. Hence, manual derivation of a supervisor for an uncontrolled system that leads to the desired behavior, especially when there is a strong interaction between systems components, is difficult and often results in supervisors that contain so-called blocking situations during operation. Blocking occurs when the system ends up in a state from which it cannot be driven towards any desired state. In particular, given the safety-critical nature of ADASs, it is essential to prevent blocking, as it might lead to hazardous situations. Due to the complexity and the large dimension of the state-space of interacting ADASs, blocking situations cannot always be overseen beforehand. In [13] and [15], the aforementioned issues were addressed by using formal specification and verification tools for debugging. However, especially for a complex system with many coupled components, this can still be a very tedious and time-consuming process. Hence, there is a strong need for (model-based) automatic synthesis tools with integrated validation and verification. For this reason, it seems more attractive to use model-based supervisor synthesis methods as discussed in [18]–[21], which have the following advantages:

(i) The synthesis procedure is based on mathematically specified requirements that only allow one interpretation.
(ii) It guarantees correctness in the sense that the resulting supervisor satisfies all the specified requirements.
(iii) It excludes blocking situations by design.
(iv) Code can be generated automatically from the resulting supervisor.
(v) It is applicable to systems with large numbers of states.

In addition, the supervisor synthesis method supports modularity of system design and requirement modeling, which eases the addition and removal of components and requirements (see e.g., [22]). As such, it is an adequate method for systematical construction of a supervisor for a complex system.

Model-based design of supervisors is based on supervisory control theory (SCT, [23]), in which system components are modeled as discrete-event systems (DESs). A DES is a state-based and event-driven system, in which the state evolves according to the occurrence of events [24]. This DES is an abstraction of the physical component, as it does not describe its continuous behavior. The desired behavior that needs to be enforced by the supervisor can be modeled by requirements. From the system model and requirement model, a supervisor is synthesized automatically, using appropriate SCT tools. This automated synthesis step ensures that the system does not violate the modeled requirements and is nonblocking.

SCT has been applied successfully in various domains, including transportation systems [25], [26] and automotive systems [27]–[29]. In [27], a supervisor is synthesized for a Cruise Control functionality for a heavy duty vehicle, in [28], a supervisor for multi-lane traffic maneuvers is considered, and in [29], a supervisor for vehicle-to-vehicle communication is designed and simulated.

In this paper, an SCT-based method is proposed for the design and implementation of supervisors coordinating multiple interacting ADASs. The design philosophy is demonstrated using a case study. A DES model representing the physical system is presented and the desired behavior is modeled by requirements. Through an automatic synthesis procedure, the supervisor is obtained. In addition to [27], the correctness of the models is assessed through simulation using various test scenarios. To evaluate the actual behavior of the system via simulation, a hybrid model is used that captures both the discrete-event and continuous-time behavior of the system. Furthermore, the code generated from the synthesized supervisor is implemented in a real vehicle and experiments are executed in open traffic, showing the effectiveness of the proposed framework. The contribution of this paper is twofold.

- It presents a systematic approach to model-based supervisor design and implementation for ADASs.
- The proposed approach is applied to a representative and illustrative case study, showing in particular:
  - Modeling of the system and requirement models;
  - Validation of models using simulation-based visualization;
  - Description of the implementation set-up;
  - Experimental validation of the implementation that has been generated for a Matlab/Simulink environment.

Additionally, all used models are made available and scripts are provided that allow to reproduce the supervisor and the implementation code [30].

This paper is organized as follows. Section II provides an introduction to the supervisor design method and SCT. Section III explains how the method is used to design a supervisor for a system with both Cruise Control and Adaptive Cruise Control. Section IV describes how the models are validated through simulations and the generated implementation is evaluated through experiments using a real vehicle system. Concluding remarks are provided in Section V.

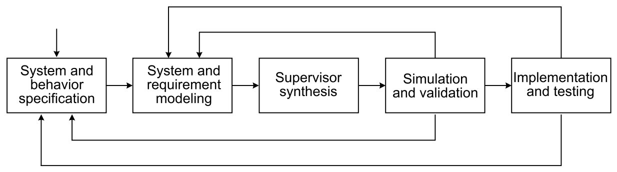
System and behavior specification
System and requirement modeling
Supervisor synthesis
Simulation and validation
Implementation and testing

Fig. 2. Model-based design method.

II. Model-Based Design Approach

The model-based approach to supervisor design that is presented in this paper consists of five steps, as illustrated in the flow diagram in Fig. 2. The goal of the method is to obtain a supervisor that restricts the uncontrolled system to the desired behavior in the sense that only events/decisions that do not lead the system to undesired states are allowed to take place. The step-by-step design method is explained below.

1) **System and behavior specification:** The first step is to analyze and specify the system components and their function. Moreover, the interaction between individual components and the desired behavior of the complete system are specified.
2) **System and requirement modeling:** Given the system specification, both a discrete-event system and a hybrid model are formalized. The DES is used for supervisor synthesis and the hybrid model is used for simulation and validation of the system behavior. For the purpose of synthesis, the desired behavior specification is translated into a set of formal requirements.
3) **Controller synthesis:** Based on the DES and the requirement model, a supervisor is synthesized using SCT. The resulting supervisor does not contain any behavior that violates the requirements or results in blocking.
4) **Simulation and validation:** The supervisor and the hybrid model can be merged to form the hybrid simulation model, which is used to validate the DES abstraction, requirements and the behavior of the system under supervision of the resulting supervisor. If the test cases reveal imperfections in either the component models or the requirement models, steps 2 and 3 can be partly repeated. Moreover, additional components and requirements can be formulated to expand the systems functionality. The advantage of validation through simulation is that it takes less effort in terms of (re-)design time in comparison to implementation and experimental testing, as shown in [31] and [32].
5) **Implementation and testing:** When the supervisor is validated in simulation, it can be implemented on the physical system and tested in real-life.

The first two steps (analysis and modeling) are manual, steps 3 through 5 (synthesis, simulation and implementation) are automatical. Steps 4 and 5 do require additional inputs that are prepared manually: representative test scenarios and hardware map, respectively, which is explained in Section II-B.

Section II-A gives an introduction to the SCT framework which is used in design steps 2 and 3. A small system is used as an example to illustrate the introduced concepts. Section II-B explains the simulation and implementation concepts that are elaborated in the case study.

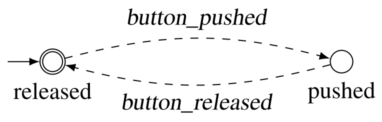

Fig. 3. Model for a button.

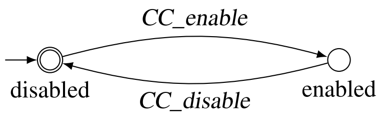

Fig. 4. Model for enabling and disabling Cruise Control.

A. Supervisory Control Theory

Supervisory Control Theory provides a framework for automatic supervisor synthesis based on models of both the uncontrolled system (also called plant) and the requirements. Most of the definitions provided here are taken from [33].

1) Specification of Uncontrolled System Behavior: In many approaches towards supervisory control theory, especially those involving computer-support (e.g., [20]), plant models are represented by so-called discrete-event systems.

Definition 1 (DES): A discrete-event system (DES) *is a quintuple* $D = (Q, \Sigma, \longrightarrow, q_0, Q_m)$ *where* Q *is a finite set of states,* Σ *is a set of events (or alphabet), partitioned into controllable events* Σ_c *and uncontrollable events* Σ_u*,* $\longrightarrow \subseteq Q \times \Sigma \times Q$ *is a transition relation,* $q_0 \in Q$ *is an initial state, and* $Q_m \subseteq Q$ *is a set of marked states.*

Controllable events can be enabled or disabled by the supervisor. Examples of controllable events are events associated with actuating the system. Uncontrollable events are events that cannot be enabled or disabled by a supervisor. Examples of uncontrollable events are changes of sensor values or buttons being pushed or released. Marked states are states associated (by the modeler) with completion of operations or tasks. The names of the states are useful for referring to states in requirement models as shown later.

A DES is typically represented in a graphical way as shown in Fig. 3. States are represented by circles and transitions are represented by event-labeled edges between two states. Transitions labeled by uncontrollable events are indicated by dashed lines and those labeled by controllable events are indicated by solid lines. The initial state is indicated by an incoming arrow. Marked states are indicated by a double circle.

The DES shown in Fig. 3 illustrates the model of a button consisting of two states, named released and pushed, and two uncontrollable transitions, labeled by *button_pushed* and *button_released*. Initially, the button is released. When a system operator pushes the button, the transition *button_pushed* is executed, after which the system is in the pushed state. The released state is chosen to be the marked state.

As another example, involving controllable events, Fig. 4 illustrates a model for the lower-level control action to enable or disable Cruise Control.

A DES is used to describe behavior, where behavior is considered a language, i.e., a set of sequences of events.

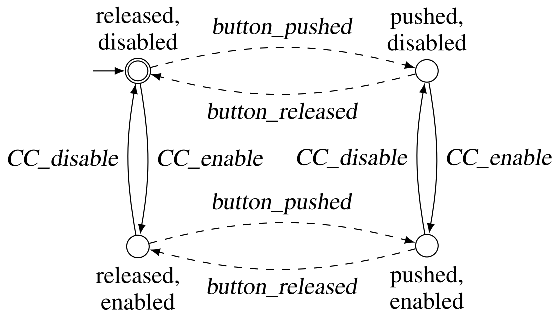

Fig. 5. Synchronous product of the models for the button and enabling and disabling CC components.

The *(marked) language* of a DES consists of all sequences of events that correspond to paths from the initial state (to a marked state of that DES, respectively). In the upcoming definitions the notation $q \xrightarrow{e} q'$ denotes $(q, e, q') \in\longrightarrow$, and for a string $w \in \Sigma^*$, $\xrightarrow{w}$ denotes the obvious generalization of the single step relation to a multi-step relation, where the events of the subsequent steps together form the string w.

Definition 2 (Behavior of a DES): The closed behavior *of a DES D is $L(D) = \{s \in \Sigma^* \mid q_0 \xrightarrow{s} q, q \in Q\}$. The* marked behavior *of D is $L_m(D) = \{s \in L(D) \mid q_0 \xrightarrow{s} q', q' \in Q_m\}$.*

It is impractical and inconvenient to specify the uncontrolled behavior of relevant systems by means of a single plant model. Usually such plant models are specified by means of networks of DESs that interact by synchronizing on their shared events (if any). In such cases, the complete plant is obtained by taking the so-called synchronous product of the DESs in the network [24].

Definition 3 (Synchronous Product): The synchronous product *of the DESs $(Q_1, \Sigma_1, \longrightarrow_1, q_{10}, Q_{1m})$ and $(Q_2, \Sigma_2, \longrightarrow_2, q_{20}, Q_{2m})$ is the DES $(Q, \Sigma, \longrightarrow, q_0, Q_m)$ where $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $\longrightarrow = \{(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2') \mid q_1 \xrightarrow{\sigma}_1 q_1',\ q_2 \xrightarrow{\sigma}_2 q_2'\} \cup \{(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2) \mid q_1 \xrightarrow{\sigma}_1 q_1'\} \cup \{(q_1, q_2) \xrightarrow{\sigma} (q_1, q_2') \mid q_2 \xrightarrow{\sigma}_2 q_2'\}$, $q_0 = (q_{10}, q_{20})$, and $Q_m = Q_{1m} \times Q_{2m}$.*

For the two example DESs shown before, which do not share any events, the result of taking the synchronous product is a DES that contains all state combinations and possible transitions. The result is depicted in Fig. 5.

2) Specification of Requirements: The second ingredient of supervisory control synthesis is a specification of the desired behavior of the controlled system, i.e., the plant under supervision of the supervisory controller to be developed.

A convenient way of specifying the desired system behavior is by posing a number of requirements that should be satisfied by the controlled system. In this paper, the following requirement specifications types are used:

1) requirement DESs, and
2) event conditions.

A requirement DES is just a DES as used in modeling the plant. It states, for the events that occur in its alphabet, in which order they are allowed to occur.

As an example, assume that the desired behavior is that any enabling or disabling of CC is supposed to occur in reaction to pushing the button. The requirement model that reflects this desired behavior is shown in Fig. 6. Note that occurrences of plant events that are not part of the alphabet of the requirement DES are not restricted (by the requirement).

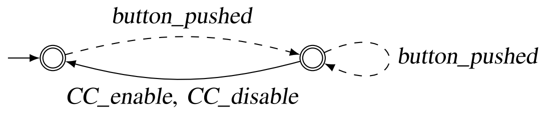

Fig. 6. Requirement model for enabling and disabling CC.

An event condition specifies that a specific event needs a specific state to be enabled. Event conditions are specific instances of so-called state-based expressions [34].

Definition 4 (Event Condition): Let e *be an event and let* Pred *be a predicate over the states of the plant. An event condition is a statement of the form "*e *needs* Pred*".*

For example, when the desired behavior is that CC can only be enabled when the button is pushed, this can be modeled using the event condition:

$$\mathit{CC\_enable} \text{ needs pushed}$$

where pushed is a reference to the state pushed of the automaton for the enable button from Fig. 3.

One can associate a requirement model with each event condition by computing the synchronous product of all DESs that are referenced in the predicate part of the event condition and removing any e-labelled transitions that start from a state that does not satisfy the state predicate, and adding an e-labelled loop in any state that does not have an outgoing e-labelled transition and satisfies the state predicate.

Definition 5 (Requirement Satisfaction): A DES D satisfies a requirement R (with alphabet Σ_R) if $P_{\Sigma_R}(L(D)) \subseteq L(R)$, where P_{Σ_R} is the projection operator that removes from all strings all events that are not in Σ_R.

3) Supervisory Control Synthesis: The purpose of supervisory control synthesis is to provide a *supervisor*, i.e., a DES that, when composed with the plant (by means of synchronous product), satisfies the requirements, and is nonblocking and controllable. Moreover, it is required that the supervisor restricts the behavior of the controlled system to the least possible extent, i.e., it is minimally restrictive.

Definition 6 (Nonblocking, Controllable, Minimally Restrictive): A DES D is nonblocking *if $\overline{L_m(D)} = L(D)$, where $\overline{L} = \{w \mid \exists_v wv \in L\}$ denotes prefix closure of a language L.*

A language $R \subseteq \Sigma^*$ is controllable *w.r.t. plant D and uncontrollable alphabet Σ_u if $\overline{R}\Sigma_u \cap L(D) \subseteq \overline{R}$.*

A DES S is called minimally restrictive *(w.r.t. DES D and language R) if $L_m(S) = \sup \mathcal{C}(R \cap L_m(D))$ where $\mathcal{C}$ is the set of all controllable sublanguages of its argument.*

The supervisor synthesis procedure computes the minimally restrictive and nonblocking supervisor, as defined above. Algorithms for supervisory control synthesis that are proven correct can be found in [24] and [35].

The supervisor that is synthesized for the plant and the requirement in Fig. 5 and Fig. 6, respectively, is represented by the DES in Fig. 7. Observe that indeed the

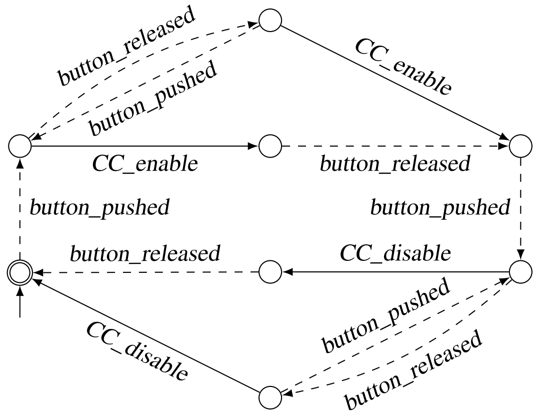

Fig. 7. Supervisor for the enable button and enabling CC.

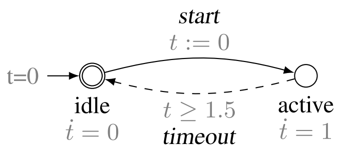

Fig. 8. Hybrid model of a timer.

supervisor complies with the requirement in the sense that *CC_enable* or *CC_disable* are not allowed before the event *button_pushed* of the button has occurred.

It is a well-known result, see, e.g., [24], that the worst-case complexity of the standard synthesis algorithm is $O(n^2 \ m^2|\Sigma|)$, where n and m are the state-space sizes of the plant and requirements representations. To cope with this complexity, several approaches have been proposed allowing for synthesis of local supervisors for plant modules. Another approach is proposed in [36], where the plant states are represented as state tree structures and symbolic computation implemented by the manipulation of binary decision diagrams (BDD) is used to synthesize the supervisor.

For the model-based design of the supervisor described in this paper, CIF 3 (`se.wtb.tue.nl`) is used [20]. The implementation of the synthesis algorithm in CIF 3 follows the BDD-based approach of [37].

B. Simulation and Implementation

For the validation of the models through simulation, a hybrid model is designed. This hybrid model consists of extended finite automata (DES extended with variables). The extended finite automata are based on the previously defined plant models, but additionally the continuous behavior is modeled.

An example of a hybrid model is shown in Fig. 8. It represents a model of a timer that consists of a continuous variable t representing passage of some time. The continuous information and the discrete information of the hybrid model are indicated with different colors. The timer can be started using a controllable event *start*, at which the variable t is updated to 0. In the active state, the derivative of t is 1. The event *timeout* is only possible at the moment the associated guard $t \geq 1.5$ evaluates to true.

The hybrid model together with the supervisor and a graphical image compose the interactive simulation model. Using test cases with predefined event sequences, the behavior of the system can be validated.

After simulation and validation, the supervisor is implemented. In the implementation, uncontrollable events are implemented as input variables. An event occurrence is associated with a value change of such a variable. Similarly, controllable events are implemented using output variables. A hardware map is designed that connects the supervisor to the hardware components of the system.

Event simultaneity as it may occur in real life is captured in the CIF models by means of interleaving, i.e., the simultaneous events may occur in any order and all these orderings lead to the same state. In the implementation, event simultaneity is supported to the extent that it is possible for input variables to change value simultaneously.

Any possible non-determinism expressed by the supervisor model is removed in the implementation. In each cycle, it is checked whether the events (both controllable and uncontrollable) are enabled in a pre-defined order. If an event is enabled, it is executed and the state change is computed and effectuated. With the resulting state the remaining (lower ordered) events are checked for enabledness. As a consequence, the event sequence that is executed by the implementation is one of the event sequences of the supervisor model.

Another consequence of this way of eliminating non-determinism is that, in general, it cannot be guaranteed that the marked states of the system are still reachable (because the events leading to them are not selected). In cases where a marked state has to be reached (which is also not guaranteed by the obtained supervisor as only the potential reachability is guaranteed) additional requirements need to be added that force the supervisor to make the required choices.

Using the CIF 3 code-generator (se.wtb.tue.nl/tools/codegen), an S-function file with C-code is generated, which may then be deployed on the actual system. The resulting S-function is implemented as part of a Matlab/Simulink model that runs with a fixed sampling rate. Of course this has as a consequence that the implementation may not notice value changes of an input variable that cancel each other if they occur within one cycle.

As an illustration of the proposed method for obtaining a correct supervisory controller for systems of ADASs, in Section III, the supervisor design for the longitudinal control of a vehicle using ADASs CC and ACC is elaborated. In Section IV the simulation, validation, implementation and experimental results are presented.

III. Case Study: Supervisor Design

This section illustrates the first steps of the supervisor design procedure as suggested in Section II. Fig. 9 shows the design elements that are considered throughout this, and the next, section. The case study is related to the supervisor design for a CC and ACC functionality for a Toyota Prius Executive. For the sake of brevity, only a part of the supervisor design is discussed. The complete model can be found in [30].

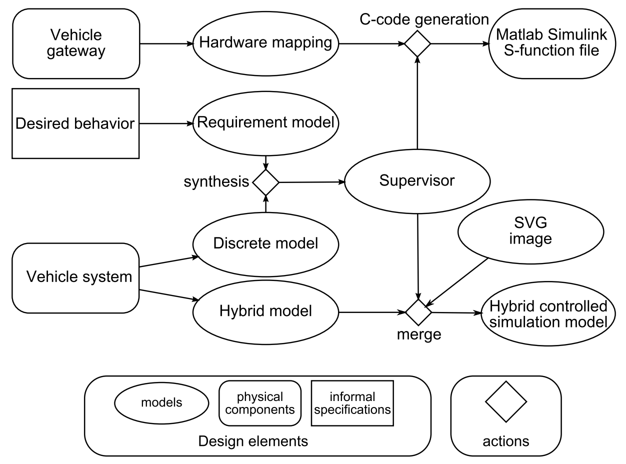

Fig. 9. Design elements of the model-based design process.

A. Specification of the System and Its Desired Behavior

As mentioned in Section II, the first step of the supervisor design process is to describe the vehicle system (components) and its desired behavior. In general, we aim to model only the system components that are needed to synthesize the supervisor. For the CC/ACC vehicle system, we only specify the set of components that are involved in the longitudinal motion of the vehicle. The system allows the driver to switch between three main control functionalities, namely Manual Control, Cruise Control and Adaptive Cruise Control. Each of these functionalities and their related system components are discussed in this subsection.

1) Manual Control: The longitudinal control of a vehicle is manually executed by the driver using the gas and the brake pedals. In modern vehicles, the depth of the pedals is read by a sensor. This sensor value is then converted into a throttle input or braking force, that is used by the throttle actuator to control the fuel supply of the engine. Manual control is required to be active at the startup of the system. Furthermore, a manual control action should always overrule the ADASs functionalities.

2) Cruise Control: Cruise Control is used to maintain a desired velocity set-point using feedback control, see, e.g., [38]. Typically, the CC generates a desired acceleration command for the vehicle driveline to maintain the desired velocity set-point. The vehicle driveline itself is assumed to be acceleration controlled, as described, for example, in [39].

The CC is operated by the driver through the HMI. The CC-related HMI components include an enable button and a multi-directional lever at the steering wheel. Via the HMI, the driver can invoke the following CC actions:

- set a new set-point velocity or resume a stored set-point velocity (if available);
- enable and disable CC;
- increase and decrease the set-point velocity;
- cancel CC (but keep the set-point velocity).

Let us remark that CC can only be active whenever it is enabled and a set speed is set. For safety reasons, additional restrictions are made to meet the safety standards defined in [40] and [41]. For example, the lower-level CC controller should be deactivated when the brake is pressed, or when the driver has overtaken the control by a throttle overrule for longer than 3 minutes. CC activation should only be allowed above a minimal cruise speed (30 km/h).

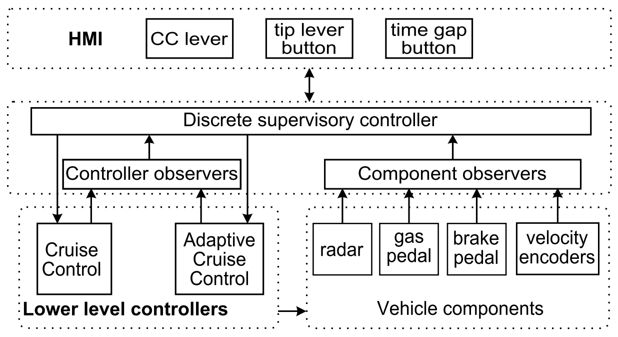

Fig. 10. System components within the controller layout of the vehicle system.

3) Adaptive Cruise Control: The ACC functionality is used to maintain a constant inter-vehicle time gap (time headway) with respect to a predecessor [42]. Therefore, ACC can only be active when another vehicle is driving in front of the ACC-equipped vehicle at an appropriate distance. The presence of a predecessor vehicle is detected by a front-facing radar, which measures the distance and the relative velocity of this predecessor. The ACC functionality can be selected by the driver using the mode button. ACC is only allowed to become active when CC is enabled, the ACC mode is selected and the radar detects another vehicle with reliable data at an appropriate distance.

B. System and Requirement Models

In this subsection, we formalize the system description provided in the previous subsection in terms of individual (discrete) component models and requirement models, see Fig. 9. As mentioned in Section II, these models are key ingredients for supervisor synthesis in step 3 of the design procedure. Moreover, hybrid versions of these models are used for validation purposes in step 4 of the design procedure.

1) System Components: The longitudinal vehicle control system consists of physical (sensor and actuator) components such as the gas pedal, brake pedal, velocity encoders and radar. Furthermore, the HMI consists of a lever at the steering wheel and an enabling button that is positioned at the tip of the lever. For ACC, an additional time gap button is present. An overview of the system components is given in Fig. 10. This figure shows how the control structure is related to the physical components, in correspondence with Fig. 1. With the vehicle components, component observers are associated. They observe the continuous state of the system and generate discrete events under relevant conditions. These events are communicated to the supervisor. The supervisor outputs the control commands that are passed to the low-level controllers. Using controller observers, also the continuous behavior of the low-level controllers is observed and communicated to the supervisor. The HMI layer communicates the driver's button push instances to the supervisor.

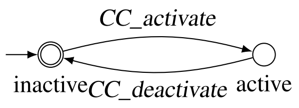

Fig. 11. Model for activating and deactivating CC.

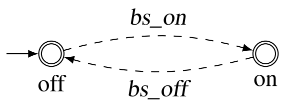

Fig. 12. Model for the brake sensor.

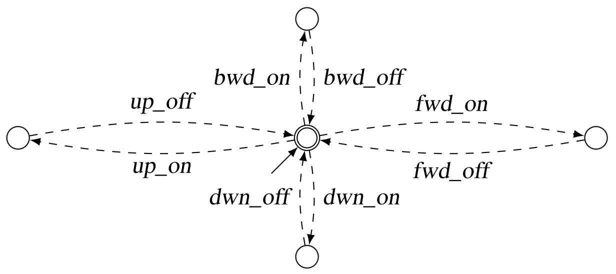

Fig. 13. Model of the CC lever.

Below, the low-level controllers, vehicle components, and controller and component observers are explained. Their relations and desired behavior are explained and their models are discussed.

2) Low-Level Controllers for ADASs: The low-level controllers are the CC controller and the ACC controller. The CC low-level controller can execute the following commands: enabling and disabling of CC, activation and deactivation of CC, the setting and resetting of the set-point velocity, increasing and decreasing of the set-point velocity and erasing the set-point velocity. Each of these control actions is modeled separately in a small, two state DES. This improves the adaptability of the model and increases the ease of adding extra control actions. Some of these DESs are shown in Fig. 4 and Fig. 11. The others are modeled similarly.

ACC can become active and inactive when CC is enabled and the driver has set the mode to CC/ACC. This activation can be modeled similarly to CC activation (Fig. 11).

3) Component Observers: To retrieve the system state, component observers are modeled. These component observers can be sensors or state observers. State observers observe the states of the vehicle components and generate events when in an important (predefined) state or when a state change occurs.

A sensor on the brake measures whether the brake is pressed or not. The sensor can either be on or off. The brake sensor automaton is depicted in Fig. 12. Note that the events that are used are uncontrollable in this case. All sensors discussed further in this subsection are modeled similarly.

A throttle overtake happens when the requested acceleration of the pedal exceeds the CC- or ACC-requested acceleration.

The velocity of the vehicle is measured by the velocity encoders on the vehicle. For each relevant velocity, a separate observer is introduced.

ACC should not be active when the CC set-point velocity is lower than the predecessor vehicle speed. Therefore, a velocity difference observer needs to be modeled.

The radar can be modeled with two states, on and off. It turns on when it detects an object and turns off when there is no object detected anymore.

The radar data is filtered to determine and increase the reliability of the signal. This is modeled using a radar reliability observer.

4) Controller Observers: Although the supervisor restricts the CC set-point velocity actions, the set-point velocity variable is modeled in the low-level controllers. Controller observers are modeled that communicate important variable changes to the supervisor. For example, since the minimum velocity for CC to become active is 30 km/h, the set-point velocity should not be able to get lower than 30 km/h. A maximum set-point velocity can be modeled similarly.

5) HMI Component Models: The HMI consists of the buttons that are used for both CC and ACC: the CC enable button, the multi-directional steering wheel lever and the ACC time gap button. The CC enable button and the ACC time gap button can be modeled as shown in Fig. 3. The model for the multi-directional steering wheel lever component is given in Fig. 13.

The ACC HMI components are two buttons. The first button is the mode button, which is a part of the multi-directional CC lever. The second HMI component is a different button positioned at the steering wheel, that can be used by the driver to set the inter-vehicle time gap.

Observe that the presented system component models do not share events. This modular and transparent system component modeling approach makes it relatively easy to incorporate additional ADASs and system components in the model.

6) CC Formal Requirements: The desired behavior of the system can be translated into a set of formal requirements. Formalized requirements are transparent and easily adaptable and expandable. Therefore, it is preferred to formulate the requirements in a short and structured way. In this paper, only a part of the formal requirements is presented. The complete model is available in [30].

7) Enabling CC: The requirements for enabling and disabling CC are elaborated in Section II-A, and the model is shown in Fig. 6.

8) Activating and Deactivating CC: CC is only allowed to become active if:

1) CC is enabled **and**
2) the velocity is (being) set **or** (velocity is set and the lever is pushed up) **and**
3) the brake sensor is not on **and**
4) the vehicle velocity is higher than 30 km/h.

CC is allowed to be deactivated if one of the following conditions hold:

1) the Cruise Control is disabled **or**
2) the brake sensor is on **or**
3) the CC lever is pulled backward (cancel) **or**
4) Manual Control (gas pedal) overtakes for longer than 3 minutes **or**
5) the vehicle velocity is smaller than 25 km/h.

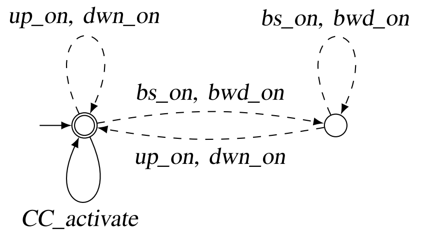

Fig. 14. Requirement model that restricts reactivation of CC.

These requirements can be modeled using event conditions as explained in Section II-A. For the sake of brevity, this is not elaborated for the requirements in this section.

These requirements alone do not lead to the complete desired behavior. When CC is active and the driver presses the brake pedal, CC is deactivated. When the brake is released however, the requirement for activating CC allows activation again. This is not the desired behavior, since CC needs to be manually reactivated. To enforce the desired behavior, a requirement model is defined that only allows reactivation after the lever is pushed up or down by the driver, see Fig. 14.

9) Setting the Set-Point Velocity: The set-point velocity can be set only if:

1) the CC lever is pushed down **and**
2) the vehicle velocity is higher than 30 km/h **and**
3) CC is enabled **and**
4) CC is inactive **or** the vehicle velocity is higher than the set-point velocity.

10) Decreasing the Set-Point Velocity: The set-point velocity can be decreased if:

1) CC is active **and**
2) the brake sensor is off **and**
3) the set-point velocity is higher than 30 km/h **and**
4) the CC lever is pushed up for more than 0.5 s **and**
5) a set-point velocity is stored **and**
6) CC is enabled **and**
7) the vehicle velocity is higher than 30 km/h.

11) Erasing the Set-Point Velocity: The set-point velocity is only allowed to be erased when:

1) CC is disabled **and**
2) a set-point velocity is stored.

12) Activating and Deactivating ACC: ACC is only allowed to become active if:

1) the mode is set to CC/ACC **and**
2) the radar is on **and**
3) the radar data is reliable **and**
4) CC is not (being) cancelled **and**
5) the brake sensor is off.

ACC is only allowed to become inactive if:

1) the mode is set to CC only **or**
2) the radar is off **or**
3) the radar data is unreliable **or**
4) CC is cancelled **or**
5) the brake sensor is on.

For ACC, a similar requirement is needed as given for CC in Fig. 14 to enforce reactivation after releasing the brake.

Modelling requirements is a manual step. In cases where the textual requirements are well-structured already, as is typically the case for safety related requirements, it is our experience that capturing these in the type of formal models used here is relatively straightforward [27], [43].

Observe that the requirements define the allowed sequences of events of the entire system. Therefore, when additional control functionalities are added, the requirement specifications may have to be changed due to inter-component dependencies. When, for example, an ADAS for lane changing is added to the complete system functionality, the requirements for CC and ACC might have to be changed.

Together, the requirement models and the event conditions (all listed in [30]) form the requirement model. This requirement model is used for supervisor synthesis, as explained briefly in the following subsection.

C. Supervisor Synthesis

When the system and requirement model are formalized, a supervisor can be synthesized that enforces the requirements and is non-blocking, controllable, and minimally restrictive, as explained in Section II-A.

For this case study, the synthesis procedure was applied to a model consisting of 28 DESs representing the uncontrolled system and 33 requirement models. All of these DES models consist of 2 or 3 states with one exception where the model consists of 5 states. The size of the uncontrolled state space is 3.4×10^9. The supervisor was obtained within 1 second (on a standard laptop). It represents a DES with a state space of 2.0×10^{10} states.

The next section describes validation of the obtained models and their implementation in a real vehicle system.

IV. Case Study: Simulation and Implementation

The previous section explains the design procedure for the supervisor. In this section, the validation through simulation of the models designed in the case study is described. After the validation, the supervisor implementation in a Toyota Prius Executive is discussed. Experimental results are provided to show the validity of the supervisor in a real-time application.

A. Model-Based Simulation

The advantage of model-based supervisor design is that simulation is easy to perform. Using different test cases it can be checked whether the models of the plant and the requirements are correct. Also environmental influences, such as the behavior and presence of a preceding car can easily be added and reproduced to test several cases.

The visualization consists of a straight driving vehicle, see Fig. 15, with buttons for the manual gas pedals and the HMI to give the driver's inputs to the supervisor.

The hybrid simulation model consists of the DES discussed before supplemented with the continuous behavior of the system components. Furthermore, the environment of the system, i.e., all objects related to the vehicle system, but not used for supervisor synthesis, needs to be modeled. This includes the gas and brake pedals, and a predecessor vehicle.

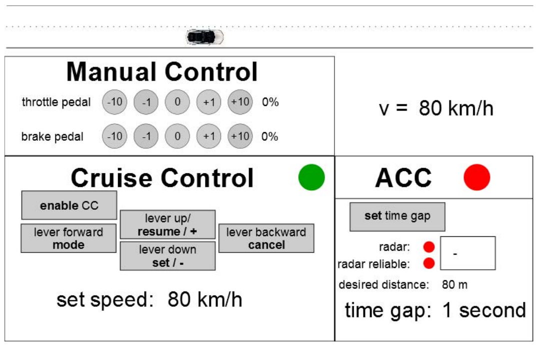

Fig. 15. Graphical representation of the simulation model.

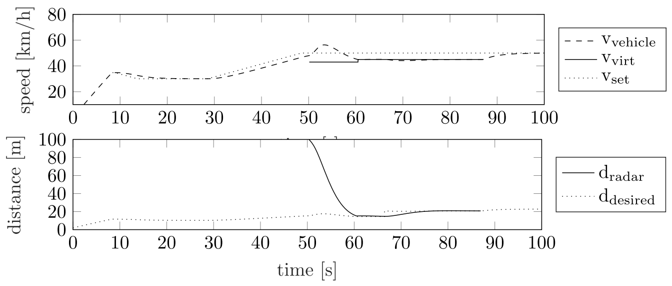

Fig. 16. Simulation results of various test cases.

For simulation, the gas and brake pedal values are modeled using a single state automaton. The values can be changed between 0 and 100 percent, see Fig. 15. The low-level controllers are designed according to [44].

Using various test scenarios that cover all of the imposed requirements we showed that the behavior of the controlled system is compliant with its expected functionality. Fig. 16 shows the simulation results of one such test scenario.

The test scenario starts with setting the throttle pedal to 80 % and enabling CC by pushing the button. As a result, CC becomes active. Then, the following actions are executed:

- To check the requirements for setting the velocity, the set velocity/- button is pushed before and after the vehicle exceeds 30 km/h. When it is pushed before, the speed is not set, but when the velocity is higher than 30 km/h, a set speed is stored. This can be seen in Fig. 16 at t = 20, where the set speed becomes approximately 34 km/h.
- To check the functionality of increasing and decreasing the set speed, the lever down button is pressed for a few seconds. Indeed the set speed decreases. At t = 30 the lever up/+ button is pressed for a while and the set speed increases as intended by the requirement.
- To check the requirement for activating ACC, it is enabled by clicking the button "lever forward". At t = 50, the radar data becomes reliable, ACC becomes active (CC becomes inactive) and the vehicle starts with distance control towards its preceding vehicle (v_{virt}).

After the models are successfully validated, the supervisor can be implemented in a real vehicle system. The implementation is explained in the following subsection.

B. Implementation

To actually check the performance of the supervisor in a real traffic environment and to check the ease of the implementation step, the supervisor is implemented in a Toyota Prius Executive (see Fig. 17). This vehicle is equipped with a dSPACE DS 1007 running a Matlab/Simulink environment, that is connected to the vehicle gateway through CAN busses [45]. Given the structural approach and using CIF 3, implementation of the supervisor is fairly simple. The supervisor is first converted into C-code and inserted in an S-function block in Simulink. The resulting S-function is implemented in a Matlab/Simulink model, that also contains the low-level controllers [45]. A hardware map is designed that connects the supervisor to the hardware components in the vehicle, possibly via observers. This S-function block is connected to the inputs and outputs of the vehicle system. The inputs and outputs are all stored as integer values 0 (false) or 1 (true). The inputs of the system are preprocessed using various Simulink blocks. The outputs of the supervisor are connected to the low-level ADASs that implement the same Simulink scheme. The experimental data is logged using ControlDesk Software.

The implemented supervisor is tested in the real vehicle system using various experiments. Again, a full validation of the supervisor is almost infeasible, as already discussed in the previous subsection. In this paper, a few experiments are discussed to show that the functioning of the controlled vehicle system is according to the required behavior. The experimental results are shown in Figs. 18 and 19.

Fig. 18 shows the experiments of the CC functionality. The top figure shows the velocity profile of the vehicle and the setpoint velocity. The bottom figure shows the driver's operation

Fig. 17. Toyota Prius Executive test vehicle.

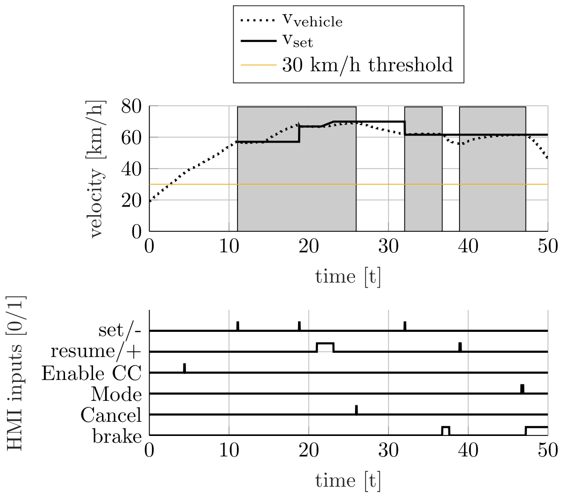

Fig. 18. Experimental results for CC. The gray parts indicate that CC is active. The bottom figure shows the HMI input and the brake sensor value.

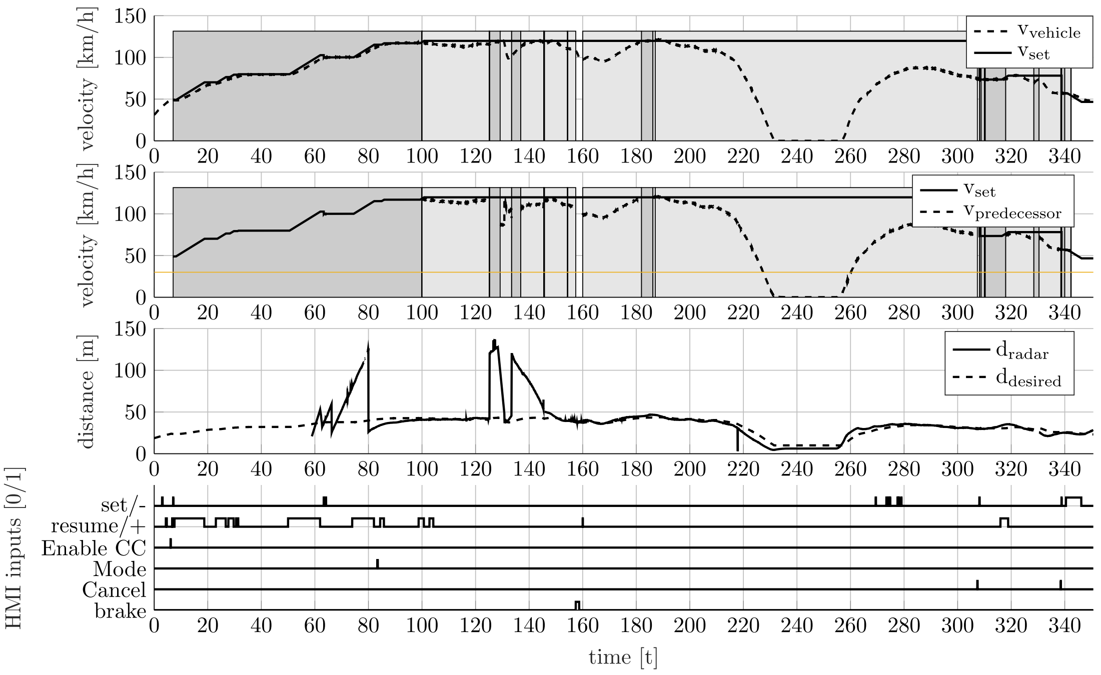

Fig. 19. Experimental results. The dark gray parts indicate that CC is active. The light gray parts indicate that ACC is active.

commands that are communicated to the supervisor through the HMI. Note that the tests are performed in a real traffic environment, and therefore many experiments are executed in parallel, whenever the traffic environment allows a specific control action from the driver. Again, we explain the validation of some functional requirements as defined in Section III-B. First, the vehicle is manually brought to a velocity of 60 km/h. The CC enable button is pushed at t = 5 to enable CC. Then, the following situations can be identified from the actions and behavior as shown in Fig. 18:

- When the CC lever is pushed in the “Cancel” direction and CC is active, CC should be deactivated. This is tested at t = 26, where CC is active (indicated by the grey block in the top figure) and the “Cancel” button is pushed (see bottom figure). As can be seen, indeed CC becomes deactivated and the vehicle slows down.
- After the CC lever is pushed in the “set/-” direction, with a speed still set (black line top figure) but with a different vehicle velocity, CC should be re-activated but with the new set speed. This test can be found at t = 32.
- When the brake is hit, CC should become deactivated. This test is executed at t = 38.
- When the CC lever is pushed in the “Resume/+” direction, CC activates, with the previous set speed (t = 40).

Fig. 19 shows the experimental results of the switching between ADASs CC and ACC. The first figure shows the vehicle velocity and the set-point velocity. The second figure shows the set-point velocity and the velocity of the preceding vehicle (if present). The third figure shows the desired inter-vehicle distance and the radar distance measurement values. The following functional checks can be identified:

- Activation of ACC when the mode is set to ACC and a preceding vehicle is detected by the radar, with lower velocity than the current set speed. At t = 83, the CC lever is pushed in the “Mode” direction. As a result (which is not shown in the picture), ACC mode is enabled. At t = 100, a preceding vehicle with lower velocity is detected (see second figure). There, ACC becomes active and CC becomes inactive. This is indicated by the light grey blocks in the top two figures.

Deactivation of ACC is tested as follows:

- At time t = 160 the brake is pressed and ACC is deactivated. After a resume however, ACC is activated again.
- At time t = 125, the preceding vehicle reaches a velocity higher than the set velocity. Therefore, ACC deactivates and CC is activated to pursue the set-speed velocity.

At the time interval from t = 220 to t = 280, a remarkable situation occurred in the test. The requirements for CC and ACC to become active both consist of the requirement that the velocity should be above 30 km/h. From time t = 220 to t = 240, the preceding vehicle slows down to a velocity of 0 km/h. As shown in the figure, ACC remains active during this period and therefore the car automatically continues following the preceding vehicle’s velocity profile. Although this behavior does not seem to be part of the desired behavior, it actually is, because the requirements only block activation of CC or ACC, not the “being active” of the functionalities.

All performed experiments show that the controlled vehicle system behaves as specified.

The supervisor is based on the discrete-event model of the physical system and since the implementation on the real physical system results in the expected functionality, the used models are a correct representation of the physical system.

V. Concluding Remarks

In this paper, a framework for systematic model-based design and implementation of supervisors for Advanced Driver

Assistance Systems (ADASs) is proposed. An advantage of this method is that it is based on formal models of both the uncontrolled system and the requirements. The fact that such models are unambiguous is instrumental as it allows (i) simulation-based validation, (ii) synthesis of a supervisor that is correct-by-construction, (iii) code generation.

A proof of concept is delivered by a realistic case study involving manual control, CC and ACC, which is implemented in a passenger vehicle and tested in a real traffic environment.

The proposed method supports incremental design in terms of changing or adding component and requirement models. Previous applications of the model-based engineering design and implementation methods in other application domains have shown that it supports the evolution of the developed controller with changes in the uncontrolled system and the requirements [31], [32]. It is expected that similar benefits can be obtained in automotive systems engineering.

For the case study, the synthesis step takes less than 1 second. For systems of more ADASs, the resource consumption of both time and memory may grow, but there are techniques available for decomposing large synthesis problems into smaller ones. Based on these observations, it is fair to conclude that the proposed method is a promising approach to the design of supervisors for systems of ADASs.

A limitation of the proposed method is the informal relationship between the models used for synthesis and for validation. Implementation of appropriate abstraction techniques would improve confidence in the results of validation.

Integration of model-based testing techniques would eliminate the need for manual construction of test scenarios.

Finally, in case a distributed or modular supervisor is created, there are still challenges with respect to its implementation such as synchronization of shared events or deployment on a single processor platform.

VI. Acknowledgment

The authors would like to thank the A-team for providing the Toyota Prius Executive as a test vehicle. Moreover, they have provided the Simulink model and vehicle gateway. The A-team is an organization consisting of students and members from Fontys Eindhoven and TU/e. The A-team is focusing on many fields of automated driving.

References

[1] Y. Ma, M. Chowdhury, A. Sadek, and M. Jeihani, “Integrated traffic and communication performance evaluation of an intelligent vehicle infrastructure integration (VII) system for online travel-time prediction,” *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 3, pp. 1369–1382, Sep. 2012.

[2] M. Mazzola, G. Schaaf, A. Stamm, and T. Kürner, “Safety-critical driver assistance over LTE: Toward centralized ACC,” *IEEE Trans. Veh. Technol.*, vol. 65, no. 12, pp. 9471–9478, Dec. 2016.

[3] A. Vahidi and A. Eskandarian, “Research advances in intelligent collision avoidance and adaptive cruise control,” *IEEE Trans. Intell. Transp. Syst.*, vol. 4, no. 3, pp. 143–153, Sep. 2003.

[4] B. van Arem, C. J. G. van Driel, and R. Visser, “The impact of cooperative adaptive cruise control on traffic-flow characteristics,” *IEEE Trans. Intell. Transp. Syst.*, vol. 7, no. 4, pp. 429–436, Dec. 2006.

[5] L. C. Davis, “Effect of adaptive cruise control systems on mixed traffic flow near an on-ramp,” *Phys. A, Stat. Mech. Appl.*, vol. 379, no. 1, pp. 274–290, 2007.

[6] F. Lamnabhi-Lagarrigue *et al.*, “Systems & Control for the future of humanity, research agenda: Current and future roles, impact and grand challenges,” *Annu. Rev. Control*, vol. 43, pp. 1–64, Apr. 2017.

[7] A. R. Girard, J. B. de Sousa, J. A. Misener, and J. K. Hedrick, “A control architecture for integrated cooperative cruise control and collision warning systems,” in *Proc. CDC*, vol. 2. Dec. 2001, pp. 1491–1496.

[8] P. Varaiya, “Smart cars on smart roads: Problems of control,” *IEEE Trans. Autom. Control*, vol. 38, no. 2, pp. 195–207, Feb. 1993.

[9] R. Horowitz and P. Varaiya, “Control design of an automated highway system,” *Proc. IEEE*, vol. 88, no. 7, pp. 913–925, Jul. 2000.

[10] S. M. Loos, A. Platzer, and L. Nistor, “Adaptive cruise control: Hybrid, distributed, and now formally verified,” in *Formal Methods*. Berlin, Germany: Springer-Verlag, 2011, pp. 42–56.

[11] S. Magdici and M. Althoff, “Adaptive cruise control with safety guarantees for autonomous vehicles,” in *Proc. IFAC World Congr.*, 2017, pp. 1–8.

[12] A. D. Ames, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs with application to adaptive cruise control,” in *Proc. CDC*, Dec. 2014, pp. 6271–6278.

[13] J. Lygeros, D. N. Godbole, and S. Sastry, “Verified hybrid controllers for automated vehicles,” *IEEE Trans. Autom. Control*, vol. 43, no. 4, pp. 522–539, Apr. 1998.

[14] P. Nilsson *et al.*, “Correct-by-construction adaptive cruise control: Two approaches,” *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 4, pp. 1294–1307, Jul. 2016.

[15] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya, “Protocol design for an automated highway system,” *Discrete Event Dyn. Syst.*, vol. 2, nos. 3–4, pp. 183–206, 1993.

[16] E. S. Kazerooni and J. Ploeg, “Interaction protocols for cooperative merging and lane reduction scenarios,” in *Proc. Int. Conf. Intell. Transp. Syst.*, Sep. 2015, pp. 1964–1970.

[17] A. Valmari, “The state explosion problem,” in *Advanced Course on Petri Nets* (Lectures on Petri Nets I: Basic Models), W. Reisig and G. Rozenberg, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 429–528.

[18] K. Chen, O. Grellier, and A. Bruyere, “Model-based control design process overview: Energetic macroscopic representation,” in *Proc. VPPC*, Oct. 2014, pp. 1–6.

[19] R. R. H. Schiffelers, R. J. M. Theunissen, D. A. van Beek, and J. E. Rooda, “Model-based engineering of supervisory controllers using CIF,” *Electron. Commun. EASST*, vol. 21, pp. 1–10, Feb. 2009.

[20] D. A. van Beek *et al.*, “CIF 3: Model-based engineering of supervisory controllers,” in *Proc. TACAS*, 2014, pp. 575–580.

[21] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, “Supremica—An integrated environment for verification, synthesis and simulation of discrete event systems,” in *Proc. WODES*, Jul. 2006, pp. 384–385.

[22] B. van der Sanden *et al.*, “Modular model-based supervisory controller design for wafer logistics in lithography machines,” in *Proc. MODELS*, Sep. 2015, pp. 416–425.

[23] P. J. Ramadge and W. M. Wonham, “Supervisory control of a class of discrete event processes,” *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[24] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY, USA: Springer, 2008.

[25] K. T. Seow and M. Pasquier, “Supervising passenger land-transport systems,” *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 3, pp. 165–176, Sep. 2004.

[26] Y. D. Yankov, “Discrete event system modeling of demand responsive transportation systems operating in real time,” Ph.D. dissertation, Univ. South Florida, Tampa, FL, USA, 2008. [Online]. Available: http://scholarcommons.usf.edu/etd/575

[27] J. M. van de Mortel-Fronczak, R. G. M. Huisman, M. H. R. van der Heijden, and M. A. Reniers, “Supervisor synthesis in model-based automotive systems engineering,” in *Proc. ICCPS*, Apr. 2014, pp. 187–198.

[28] G. von Bochmann, M. Hilscher, S. Linker, and E.-R. Olderog, “Synthesizing controllers for multi-lane traffic maneuvers,” in *Dependable Software Engineering: Theories, Tools, and Applications*. Cham, Switzerland: Springer, 2015, pp. 71–86.

[29] A. Sharma and M. Reniers, “Integrated simulation of CIF3 and Simulink models,” in *Proc. ITSLE*, 2016, pp. 33–37.

[30] T. Korssen, “Systematic model-based design and implementation of supervisors for advanced driver assistance systems,” M.S. thesis, Dept. Mech. Eng., Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2017. [Online]. Available: https://cifsupervisorycontrol.files.wordpress.com/2017/01/masters-thesis-tim-korssen.pdf

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

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

[33] K. Cai and W. M. Wonham, "New results on supervisor localization, with case studies," *Discrete Event Dyn. Syst.*, vol. 25, nos. 1–2, pp. 203–226, 2015.

[34] 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 *Proc. CDC*, Dec. 2010, pp. 3481–3486.

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

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

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

[38] A. G. Ulsoy, H. Peng, and M. Çakmakci, *Automotive Control Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[39] D. N. Godbole and J. Lygeros, "Longitudinal control of the lead car of a platoon," *IEEE Trans. Veh. Technol.*, vol. 43, no. 4, pp. 1125–1135, Nov. 1994.

[40] *ISO 26262 Road Vehicles Functional Safety*, Int. Org. Standardization, Geneva, Switzerland, 2011.

[41] *Adaptive Cruise Control System Overview*, U.S. Softw. Syst. Saf. Work. Group, Anaheim, CA, USA, Apr. 2005.

[42] R. Rajamani, "Adaptive cruise control," in *Encyclopedia of Systems and Control*. Boston, MA, USA: Springer, 2015, pp. 13–19.

[43] F. F. H. Reijnen, M. A. Goorden, J. M. van der Mortel-Fronczak, and J. E. A. Rooda, "Supervisory control synthesis for a waterway lock," in *Proc. CCTA*, Aug. 2017, pp. 1562–1568.

[44] J. Ploeg, N. van de Wouw, and H. Nijmeijer, "L_p string stability of cascaded systems: Application to vehicle platooning," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 2, pp. 786–793, Mar. 2014.

[45] V. S. Dolk *et al.*, "Cooperative automated driving for various traffic scenarios: Experimental validation in the GCDC 2016," *IEEE Trans. Intell. Transp. Syst.*, to be published. [Online]. Available: http://ieeexplore.ieee.org/document/8074808/

Tim Korssen received the M.Sc. degree in mechanical engineering from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2017.

His research interests include model-based supervisor design, discrete-event systems, and intelligent transportation systems.

Victor Dolk received the M.Sc. degree (*cum laude*) in mechanical engineering from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2013, and the Ph.D. degree *(cum laude)* in 2017.

His research interests include hybrid dynamical systems, networked control systems, intelligent transport systems, and event-triggered control.

Joanna van de Mortel-Fronczak received the M.Sc. degree in computer science from AGH University of Science and Technology, Cracow, Poland, in 1982 and the Ph.D. degree in computer science from Eindhoven University of Technology, Eindhoven, The Netherlands, in 1993.

Since 1997, she has been with the Department of Mechanical Engineering, Eindhoven University of Technology. Her research interests include model-based engineering and the synthesis of supervisory control systems.

Michel Reniers (S'17) is currently an Associate Professor in model-based engineering of supervisory control at the Department of Mechanical Engineering, Eindhoven University of Technology. He has authored over 100 journal and conference papers, and is the supervisor of ten Ph.D. students. His research portfolio ranges from model-based systems engineering and model-based validation and testing to novel approaches for supervisory control synthesis. Applications of this work are mostly in the areas of high-tech systems and cyber-physical systems.

Maurice Heemels (F'16) has held Visiting Professor positions at the Swiss Federal Institute of Technology, Switzerland, in 2001, and at University of California at Santa Barbara in 2008. In 2004, he was with Océ, The Netherlands. He is currently a Full Professor with the Department of Mechanical Engineering, Eindhoven University of Technology. His current research interests include hybrid and cyber-physical systems, networked and event-triggered control systems, and constrained systems, including model predictive control. He served/s on the editorial boards of *Automatica*, *Nonlinear Analysis: Hybrid Systems*, *Annual Reviews in Control*, and IEEE TRANSACTIONS ON AUTOMATIC CONTROL. He was a recipient of a personal VICI grant awarded by STW (Dutch Technology Foundation).