Controlling Concurrent Change – A Multiview Approach Toward Updatable Vehicle Automation Systems

Mischa Möstl
Technische Universität Braunschweig, Institute of Computer and Network Engineering
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
moestl@ida.ing.tu-bs.de

Marcus Nolte
Technische Universität Braunschweig, Institute of Computer and Network Engineering
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
nolte@ifr.ing.tu-bs.de

Johannes Schlatow
Technische Universität Braunschweig, Institute of Computer and Network Engineering
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
schlatow@ida.ing.tu-bs.de

Rolf Ernst
Technische Universität Braunschweig, Institute of Computer and Network Engineering
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
ernst@ida.ing.tu-bs.de

Abstract

The development of SAE Level 3+ vehicles [24] poses new challenges not only for the functional development, but also for design and development processes. Such systems consist of a growing number of interconnected functional, as well as hardware and software components, making safety design increasingly difficult. In order to cope with emergent behavior at the vehicle level, thorough systems engineering becomes a key requirement, which enables traceability between different design viewpoints. Ensuring traceability is a key factor towards an efficient validation and verification of such systems. Formal models can in turn assist in keeping track of how the different viewpoints relate to each other and how the interplay of components affects the overall system behavior. Based on experience from the project Controlling Concurrent Change, this paper presents an approach towards model-based integration and verification of a cause effect chain for a component-based vehicle automation system. It reasons on a cross-layer model of the resulting system, which covers necessary aspects of a design in individual architectural views, e.g. safety and timing. In the synthesis stage of integration, our approach is capable of inserting enforcement mechanisms into the design to ensure adherence to the model. We present a use case description for an environment perception system, starting with a functional architecture, which is the basis for componentization of the cause effect chain. By tying the vehicle architecture to the cross-layer integration model, we are able to map the reasoning done during verification to vehicle behavior.

2012 ACM Subject Classification Hardware → Safety critical systems

Keywords and phrases safety, behavior, functional, architecture, multi-view, automated driving

Digital Object Identifier 10.4230/OASIcs.ASD.2019.4

Funding This work was funded by the Deutsche Forschungsgemeinschaft (DFG) under the grant FOR1800 Controlling Concurrent Change (CCC).

Acknowledgements The authors thank the other members of the CCC research group for many fruitful discussions and their support.

1 Introduction

In recent years, huge progress has been generated toward the commercialization of automated vehicles systems. The focus of the industry has shifted from advanced driver assistance systems (ADAS), corresponding to SAE level 1 and 2 [24] to automated vehicle systems of SAE Levels 3+. However, while impressive results are achieved regarding environment perception algorithms, also due to the introduction of machine learning technology, verification and validation of Level 3+ systems becomes increasingly difficult. This is especially true, if it must be considered that software intense systems will most likely require frequent after-market updates for deploying bugfixes, and/or updates of the vehicle's functionality.

Challenges for safety verification are on the other hand caused by increased complexity of the perception systems required to generate a representation of the vehicle's environment which is sufficiently detailed to make decisions in complex traffic scenes (cf. [13]). On the other hand, replacing the driver is equivalent to replacing vast parts of the safety system of SAE Level 1 and 2 systems. Established safety design processes must thus be rethought and extended in order to suit the newly arising challenges when removing the driver from the control loop. Safety strategies which only assure that the driver can control system failures by being able to physically overrule system commands to the drive train or steering system do not apply anymore.

For the automotive industry, the safety standard ISO 26262 [10] provides guidelines for designing functionally safe systems. This subsumes hazards caused by malfunctioning behavior of E/E components and ensures the correct implementation of functional (safety) requirements. One frequently formulated drawback of the ISO regarding the applicability to Level3+ systems is that it does not consider nominal behavior of the overall E/E system (cf. [19, 4, 13, 8]) and thus does not provide guidelines on how to define the functional requirements for the system. However, this formulation of safe nominal behavior (or *external behavior* as defined in [20], according to [3]) and the boundaries of safe nominal behavior is crucial when it comes to ensuring safety of driverless vehicles, as the system must not pose a threat to its passengers and/or other traffic participants. For this publication, we adopt the terminology as defined by Waymo in their 2017 safety report [32], referring to the process of defining safe nominal behavior as *behavioral safety* (cf. [4]). The upcoming ISO PAS 21448 "Road Vehicles – Safety of the Intended Functionality" is partially addressing this problem, however the scope of the current draft standard is intentionally limited to SAE Level 1 and 2 systems [12], while the defined concepts might also apply to levels of higher automation.

While there is a number of recent publications on how to extend the concept phase of ISO 26262 toward the definition of safe behavior [19, 4, 8], e.g. based on a scenario-driven concept phase, we would like to elaborate on the consequences of behavioral safety considerations from a systems engineering point of view. As we have argued in [4], the design of safe automated vehicle should follow a *safety by design* paradigm as a cross-domain effort over different disciplines. For this purpose we have proposed an architecture framework in accordance with ISO 42010 [11], featuring safety as a cross-cutting viewpoint and formulating a *functional*, a *capability*, *software* and *hardware* viewpoint and attributing *behavioral safety* to the former two and *functional safety* to the latter two viewpoints. Correspondences and correspondence rules, as defined in ISO 42010, are represented in example mappings between components in the respective viewpoints. While we formulate the need for formal methods to represent and instantiate the different viewpoints in the architecture framework, the actual instantiation was not part of the initial contribution.

Behavioral and functional safety of vehicle automation systems is one of the grand challenges for future automotive systems. Reducing the necessary testing efforts to ship updates for vehicular systems, especially of models that are already in production and in the field is the second grand challenge. The later is particularly interesting to reduce costs. In this paper, we want to show that concepts for safety related systems engineering ([4]) can be combined with automated integration mechanims and tools as investigated in the project *Controlling Concurrent Change (CCC)*[1]. As a result of such a design and integration flow, we envision systems where software updates and upgrades can be easily deployed at a minimum of cost for integration testing and safety validation through testing.

We illustrate this idea based on an update scenario for the automation system of a research vehicle. Therefore, we first introduce the architecture framework we use to asses behavioral safety in section 2. We maintain traceability from the functional viewpoint up until integration in this architecture (cf. Figure 2), by using Traceability in this architecture is maintained in two ways: For behavioral safety the process is still manual, first ideas to further automate this are also presented in section 2. The example showcase is then presented in section 3, while section 4 then presents the key ideas how we automate the integration and verification based on the presented architecture framework. This section also includes a description of the resulting cross-layer system model. Finally, section 5 concludes the paper.

2 Behavioral Safety in Systems Engineering

As stated in Section 1, the concept of behavioral safety is a potential missing link to extend the concept phase of established ISO-26262-compliant processes toward the application for SAE Level 3+ vehicle automation systems. In this section, we summarize the architecture framework described in [4] and discuss the implications of behavioral safety on traceability requirements for system properties in the design phase and at runtime.

Considering behavioral safety as an integral part of the safety concept creates the problem of defining appropriate behavior in different scenarios [4]. An example scenario is displayed in Figure 1 with the vehicle approaching a pedestrian crosswalk.

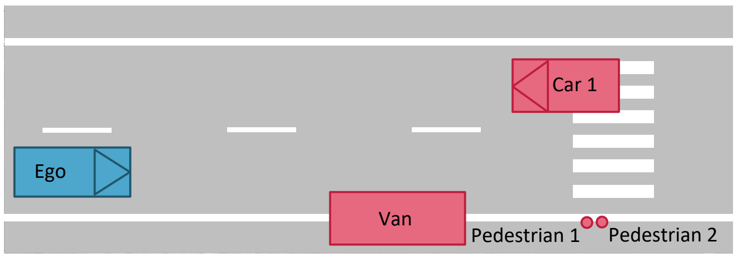


Figure 1 Example scenario: Automated vehicle approaching a pedestrian crosswalk occluded by a parked vehicle with oncoming traffic and pedestrians who are likely to cross.

At the scenario level, abstract safety goals can be formulated, e.g. by stating that the automated vehicle must not enter oncoming traffic. A process of how these abstract safety goals can be decomposed into (functional) safety requirements and actual technical requirements has been formulated in [4]. A short summary of the described process following an (iterative) Item Definition can be stated as follows:

[1] https://ccc-project.org

1. Conduct Hazard Analysis and Risk Assessment by possible accidents in the defined scenarios.
2. Define safety goals.
3. Define a risk minimal state for the scenario at hand.
4. Define functional safety concept (safety requirements and hazard mitigation strategies) for fulfilling the safety goals.
5. Combine with functional architecture and required system capabilities derived during the item definition to derive technical requirements.

However, the consequences of formulating behavioral safety requirements for systems development reaches further than defining requirements at the beginning of the development and validation and test before market release in a classic V-Model-like development process. In addition, the adherence to safety requirements must be monitored at runtime. This is required to initiate emergency strategies for reaching a risk minimal state in case safety requirements are violated.

For monitoring system behavior at runtime, we have proposed the application of ability and skill graphs [21] and their integration into a development process [20]. They represent functional dependencies in the system, formulating the required capabilities to fulfill the vehicle's mission. They explicitly model external system behavior as well as dependencies for performance assessment at a functional level, and provide guidance for the decomposition of functional requirements into technical requirements.

A core question which needs to be addressed in this context is, how the technical implementation, which is subject to functional safety requirements can cause hazards at the behavioral level. This is where traceability aspects come into play: Assuming that a *functional* system architecture and a capability representation are available after the concept phase of the development process, technical architectures in terms of hardware and software architectures are developed during system implementation. As defined in the ISO 42010, a sound architecture framework requires the formulation of correspondences and correspondence rules between different architectural views. In our formulated architecture framework (cf. Figure 2), this means that we need mapping relations between *functional*, *capability*, *hardware* and *software* components (depicted in Figure 2 as red arrows).

However, while informal formulations of those correspondences can assist during system development, informal notations are not suitable to support system monitoring at runtime. For this purpose a formal system model is required which can relate formalized requirements to the current system configuration, e.g. including component mappings or interface and task dependencies.

To demonstrate this, we performed a manual ability and skill graph based assesment for an automated driving function of a research vehicle, and used its results as the input for a model-based integration flow. The vehicle and the automation function is explained in the following section. How this function is integrated into a vehicle system in a correct-by-construction fashion is subsequently explained in section 4, which will explain our cross-layer model instantiating the multi-view architecture.

3 Concurrent Change Use-Case

The CCC approach combines a conventional lab-based design of individual functions with an automated integration process which ensures that updates are applied to an already deployed system only if the system can still adhere to the required safety and security constraints.

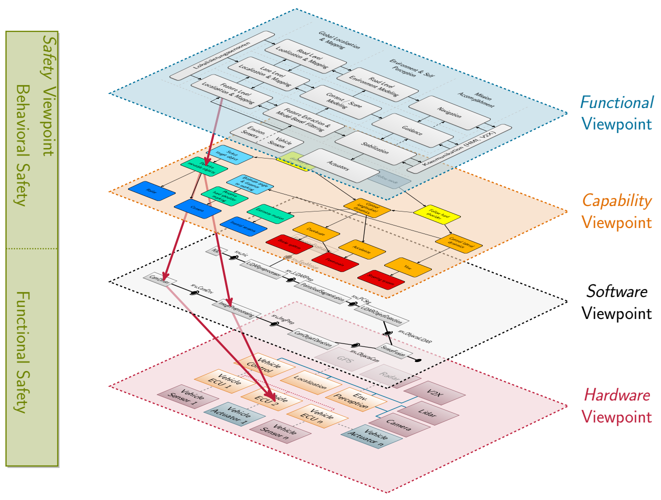


Figure 2 Architecture framework presented in [4]: Showing safety as a cross-cutting viewpoint orthogonal to the functional, capability, hardware and software viewpoint. Note that depicted architectures are examples, such as a component architecture in the software viewpoint. Red arrows depict example mapping relations (correspondences) between components in different viewpoints.

This becomes particularly challenging as the target platform is shared by multiple functions with different criticality. All side effects must therefore be anticipated and either be bounded or mitigated in order to ensure safe operation of critical functions at all times. For this purpose, all requirements and constraints must be explicitly specified in the input models. Another challenge consists in finding (and specifying) appropriate abstractions that guide the decisions which must be made during such a model-based integration process, as these are usually based on experience and expert knowledge, which is only implicitly available.

We demonstrate the applicability of the approach on an environment perception and motion planing showcase that we will introduce in the following:

3.1 Research Vehicle MOBILE

For showing the applicability of the approaches developed in the CCC project in an automotive context, the research vehicle MOBILE [5] built at the Institute of Control Engineering at TU Braunschweig serves as a demonstrator platform.

MOBILE was originally built as a demonstrator for the development of vehicle dynamics control algorithms and vehicle systems engineering applications. It features of four close-to-wheel electric drives ($4 \times 100\,\mathrm{kW}$), as well as individually steerable wheels, and electro-mechanic brakes [5]. The vehicle features a FlexRay backbone for inter-ECU-communication and additional CAN bus interfaces, which are used for communication with sensors and actuators for vehicle control. The ECUs for vehicle control are programmed in a customized MATLAB/Simulink tool chain. Combined with detailed vehicle-dynamics models, the tool chain serves as a means to establish a rapid-prototyping process for vehicle control algorithms.

The basic idea in the project scope is to demonstrate how the CCC architecture can contribute to a state-of-the-art environment perception system in an automated vehicle. For this purpose, the research vehicle MOBILE has been equipped with three roof-mounted LiDAR sensors (cf. Figure 4a), as well as a highly accurate localization platform. Additional hardware platforms were installed in the vehicle to run environment perception and motion planning algorithms in the CCC middleware. The ECUs and sensors of the CCC subsystem are interconnected by Ethernet and connected to the legacy vehicle control through a CAN interface. In addition, the algorithms can be run on a legacy platform as it is used in the *Stadtpilot* [33] project for comparison.

3.2 Environment Perception & Trajectory Planning System

The sensors provide a 360° representation of the vehicles' environment and enable it to navigate its path around obstacles in its vicinity. For the CCC project, we have ported selected algorithms from the *Stadtpilot* project, focusing on the representation of the static vehicle environment. For this purpose, incoming sensor raw data from the LiDAR sensors is combined into a point cloud. Each measurement contains position and reflectivity information. Thus, apart from the information about obstacle positions, reflectivity information can be used to create a monochrome image, making it possible to e.g. detect lane markings in the LiDAR data.

In several steps, measurements are annotated with measurement classes (e.g. ground measurements, valid measurements on actual objects, clutter, etc.). The resulting annotated point cloud is then fed into an occupancy grid [6] (cf. Figure 3a), which accumulates measurements over time. The grid framework is based on a multi-layer approach to represent environment features in distinct layers. Examples of three layers are depicted in Figure 3. Figure 3a shows an example of occupancy information in terms of free (green), occupied (red) and unkown (dark blue) space. In addition, the mentioned reflectance information (Figure 3b) for ground-labeled points is represented in a separate layer. For a more detailed description of the processing chain, please refer to [14], [23]. At the end of the sensor-data

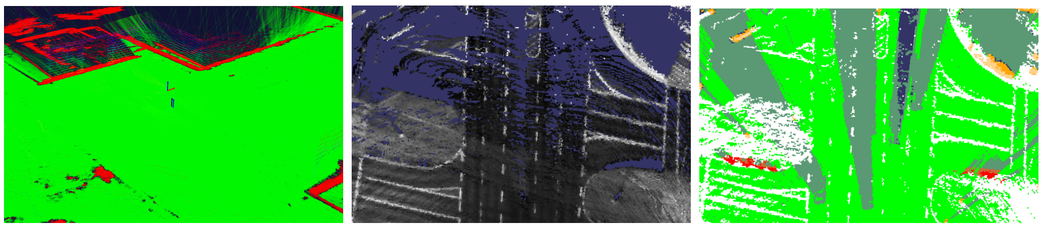

(a) Occupancy grid: discretized map displaying free (green) and occupied areas (red) around the vehicle. **(b)** Reflectance grid: reflectance values allow detecting lane markings (white) [23]. **(c)** Fused grid layer: each color indicates a different represented feature [23].

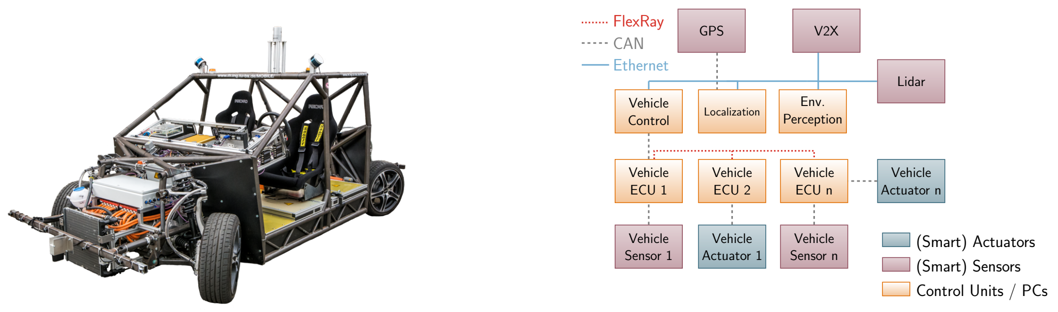
Figure 3 Three layers of the grid framework to represent environment features on an intersection.

processing chain for the static environment, the different layers are fused into a consistent representation of the static vehicle environment.

The grid representation is always kept in a local coordinate frame, which moves with the vehicle. The vehicle's position is acquired from an accurate tightly-coupled GNSS/INS platform (global position is obtained via GPS and fused with accelerations & angular rates).

(A) fused local occupancy grid(s) provide the basis to perform trajectory planning for automated driving. For this purpose, the system generates a target pose in a reachable area of the vehicle's environment and the trajectory is planned from the current position to the target pose in the vehicle coordinate frame. Trajectory planning is performed in a model-based fashion, using front- and rear-axle steering. The underlying trajectory control algorithms use the available actuators (4× steering, 4× brakes & drives) to control the vehicle to the planned trajectory. For details on and architectural considerations for trajectory planning, refer to [19]. Aspects of the applied control algorithms are presented in [30].

By representing the available actuators, trajectory planning considers the vehicle's current abilities. By monitoring e.g. sensor quality, actuator performance and control quality, the system will be able to react to failures in the system. A simple example here is the presence of a steering actuator failure, which can be compensated at the control level, as well as by adapting the trajectory planning algorithm. Monitoring of non-functional properties, such as timing is performed directly in the middleware.



(a) Roof-mounted lidar sensors. **(b)** Hardware architecture used in MOBILE.

Figure 4 Research vehicle MOBILE.

The algorithms required to demonstrate the use case will run in a distributed system, as shown in Figure 4b. The platform can be separated into two parts: While the lower part of the displayed ECUs is responsible for controlling the individual actuators of the vehicle, the upper part performs environment perception and trajectory planning tasks. As the CCC middleware only runs in the context of the environment perception system, the system model must support transitions between legacy-parts of the system, running without the project middleware and those parts, which are fully controllable by the Multi Change Controller (cf. section 4).

A coarse grained functional architecture of the use-case is depicted in Figure 5.

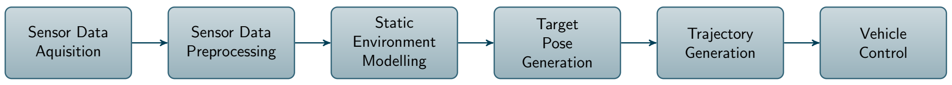


Figure 5 Coarse functional system architecture.

4 CCC's integration and verification system

For the model-based integration approach pursued here, the system is composed of two segregated domains: the model domain and the execution domain.

Figure 6 shows the conceptual setup of the system. A Multi-Change Controller (MCC) (red) hosts the model domain, consisting of the cross-layer model, as well as configuration generation and verification. We aim for component-based models – including software as well as hardware components – as they reduce dependencies in the architecture to the explicitly modeled interfaces. The components are generic building blocks of the system that is composed of these components such that they implement the desired functionality and fit to the particular target platform. Each change to the system must be coherently representable in this system-wide model for analyzing any potential cross-layer dependencies, as well as for other analyses to ensure freedom from interference for the individual functions that a set of (software) components create.

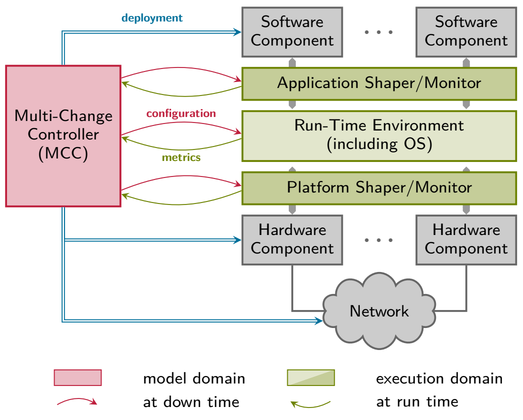


Figure 6 CCC architecture comprising a model domain (red), an execution domain (green) as well as changing software/hardware components (gray).

Similar to the conventional V-model development process, the MCC gradually refines the model representation of the new system configuration during the integration process. This is done based on a cross-layer model that captures relevant viewpoints of the system. The process generates new configurations and subsequently checks them for requirements satisfaction. If a new configuration satisfies all requirements and is rated as an improvement to the current one, it can be deployed into the execution domain.

Verification is separated due to the fact that not all requirements can be systematically considered during configuration generation. E.g. software response times are hard to optimize if arbitrary activation patterns are assumed. Consequently, an autonomous configuration and verification goes beyond a multi-dimensional optimization of requirement satisfaction.

Our execution domain, is based on the open-source *Genode OS Framework* [7]. This framework follows the microkernel approach and employs a strict decomposition of the system on the application level, resulting in a service-oriented architecture in which separate components implement and provide services for other components. While decomposition can already deal with liveliness issues [1] that arise in mixed-critical systems, dependencies on the execution time or response time of other components remain. Note that, however, the methods developed in the model domain are not restricted to these semantics but can be adapted to different implementation models.

4.1 The MCC's cross-layer model

The core concept of the MCC's model domain is that a) the system is represented on different layers of abstraction, and b) that models describing different viewpoints of the architecture are connected through mappings. Consequently, the described mappings between model artifacts are the implementation of corespondences from the abstract architecture framework.

To perform the integration task in the MCC we define three architecture layers, where each layer is treated as a graph. The top layer is a *function model*, that captures functional aspects implementation independent.

▶ **Definition 1.** *A function model is a graph* $\mathcal{FG} = (F, \hookrightarrow)$ *where the nodes in* F *describe the functions, and* $\hookrightarrow$ *is the set of edges that describe functional interactions.*

For instance the function chain depicted in Figure 5 fulfills this definition.

A further necessity of such a layer lies in the fact that safety requirements are derived from implementation independent functional descriptions of a system [10, Part1] (cf. section 2). In order to implement functions, they are decomposed into components. Since during implementation, mappings of software components to hardware components might already be fixed, e.g. because code of one software component requires certain peripherals of a hardware component, they are already part of the component model. In our employed Run Time Environment (RTE), data exchange from one component to one or more others is performed through read-only memory (ROM) components. ROMs implement synchronous bulk transfer of data based on remote procedure calls (RPCs) [7]. If a reader on a remote resource requires contents of a ROM, proxy ROM components on both ends of the communication are inserted that provide the required data on the remote side via a network connection. Formally we define:

▶ **Definition 2.** *A component model is a graph* $\mathcal{CG} = (C \cup \mathcal{C}_{Roms} \cup \mathcal{R}^{abs}, \xrightarrow{w} \cup \xrightarrow{r} \cup \xrightarrow{m})$ *where the nodes are the unified set consisting of* C *that describes the set of software components implementing functions,* $\mathcal{C}_{Roms}$ *the set of ROM components, and* $\mathcal{R}^{abs}$ *the set of abstract resources of the system. The edges either describe a read* ($\xrightarrow{r}$) *or write* ($\xrightarrow{w}$) *operation between software components and ROMs, or a mapping* ($\xrightarrow{m}$) *of a software component to a resource* ($\mathcal{R}^{abs}$)*.*

In the course of generating configuration candidates the MCC applies pattern based transformations on $\mathcal{FG}$ to produce a component model instance $\mathcal{CG}$. For the example use-case the function chain from Figure 5 is mapped to components in Figure 7 (second layer from the top). The transformation is based on selecting components that implement a function from a component repository. The repository is populated through formal xml-based descriptions of components. A more detailed account of this transformation is provided in [28].

In a subsequent step, the MCC's configuration generation refines the component model to an instance model, which only contains instantiated components. This process also allows refining components $c \in C$ into sub-components, which again can be linked by ROM components. The semantics of the resulting instance model are similar to $\mathcal{CG}$, however it only contains the minimal number of component instantiations under cardinality constraints, i.e. the maximum number of instantiations of a component on a particular CPU. This also results in a mapping of components to particular hardware components, i.e. from abstract resources to individual CPUs. The instance model of the use-case is depicted as the third layer from the top in Figure 7. Yet note, that some components are shown as composites (light blue) due to space limitations.

The knowledge of the concrete instance model together with the knowledge about the communication mechanisms allows the MCC to derive and map additional layers that model certain aspects of the system in order to represent particular viewpoints such as safety, availability or security. The requirements for these viewpoints – e.g. a safety-level requirement or a real-time constraint – are collected for each component in a so-called contracting language, which serves as an input to the MCC. Viewpoint-specific analyses are implemented as separate entities in the MCC, e.g. in order to resolve run-time dependencies between software components as presented in [27].

4.2 Analysis and Verification by the MCC

For this paper we restrict the scope to outlining how timing and safety requirements are verified by the MCC. W.r.t. safety we further limit ourselves to freedom from timing interference. For the external behavior of the vehicle, timing properties are crucial when it comes to vehicle control. As unaccounted delays can cause degraded control performance or even instable controllers, the adherence to timing constraints in the timing domain must be ensured.

In order to reason about end-to-end function timing, a model describing the timing behaviour is necessary. The transformation of the component-based software structure of the Genode OS Framework together with the RPC semantic used by the ROM components to a timing model is described in detail in [26]. The transformation result explicitly expresses effects such as blocking and priority inheritance, while preserving the event chains. [26] also describes how response-time bounds can be computed over a chain of components. Possible alternatives to compute response-time bounds is e.g. MAST [2].

However, if hardware resources are shared with components from other cause effect chains, possibly even components with a different criticality than the chain under analysis, only verifying response-time bounds is insufficient. In the use-case depicted in Figure 7 this is the case for the shared vehicle network. Following a conservative design strategy, a designer would have to assume that by sharing the resource the components are mutually dependent and that any dependency leads to interference, i.e. failures causing malfunctioning behavior. Consequently, absence or strict bounds on the dependencies have to be proven in order to argue freedom from interference.

A timing model for the Ethernet network can be derived from the knowledge of: (i) how traffic is routed through the network, (ii) which components inject Ethernet frames into the network at which rate, and (iii) what the maximum payload per frame is.

How traffic is routed is known, as this is under control of the MCC which also deploys the network configuration. Similarly, the components which inject frames are already known in the component model. The rate at which they emit a frame i into the network can be abstracted by standard event models, δ_i^+, δ_i^-. These are event model abstractions of concrete execution traces that capture the maximum/minimum time interval between n consecutive activation events. δ_i^+ and δ_i^- for a frame i can be derived from the results of the timing analysis of the component chains on the computation resources with the analysis described in [26]. Only the maximum payload per frame must be extracted from contracting information, which must be fed into the MCC. Based on this information a timing model for the Ethernet network as e.g. described in [31] can be derived. It is formally based on Compositional Performance Analysis (CPA) [22]. In this model each task τ_i represents a frame that is competing for arbitration on a switch port, i.e. the switch ports are the resources. The payload of each frame is captured by bounds on its worst-case execution time (WCET) C_i^+/best-case execution time (BCET) C_i^- on the wire including all protocol overhead. Chains of dependent tasks on different resources, i.e. Ethernet switch ports, then model a data stream. This model provides the basis to derive the timing-dependency graph (TDG) for the network and the components injecting the traffic as e.g. described by [15].

▶ **Definition 3.** *A Timing Dependence Graph is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of nodes $v_i, v_j \in \mathcal{V}$ and edges $e_k \in \mathcal{E}$ where each edge $e_k = (v_i, v_j)$ describes that v_j is dependent on v_i. Each node v_i either describes a task parameter $p \in P = \{C^+, C^-, \delta_{in}^+, \delta_{in}^-\}$ or an (intermediate) timing analysis result $r \in R = \{w^+, w^-, \delta_{out}^+, \delta_{out}^-, R^+, R^-, q_{max}\}$.*

To transform the timing model's parameter and results into a TDG, two conversion functions are necessary to populate the edge set of the TDG $\mathcal{G}$.

▶ **Definition 4.** *The parameter conversion function is a function*

$$\vartheta_p : \mathcal{T} \times \{C^+, C^-, \delta_{in}^+, \delta_{in}^-\} \mapsto \mathcal{V} \qquad (1)$$

that maps each input parameter type $p \in P$ for a task $\tau_i \in \mathcal{T}$ to a node $v = \vartheta_p(\tau_i, p)$ with $v \in \mathcal{V}$ in the TDG, and the result conversion function:

$$\vartheta_r : \mathcal{T} \times \{w^+, w^-, \delta^+_{out}, \delta^-_{out}, R^+, R^-, q_{max}\} \mapsto \mathcal{V} \tag{2}$$

that maps each result type $r \in R$ of a task $\tau_i \in \mathcal{T}$ to a node $v = \vartheta_r(\tau_i, r)$ with $v \in \mathcal{V}$ in the TDG.

This conversion function is analysis-specific, i.e. how CPA's busy-window (w^+/w^-) and output event models ($\delta^-_{out}/\delta^+_{out}$) are computed. In general, a TDG is constructed in four steps: First, for each task in the task graph, the timing dependency graph is populated with the nodes describing its parameters. In the second step, all explicit dependencies between tasks on different resources are added as edges in the graph. This happens for two tasks τ_a and τ_b by inserting two edges $e_k = (v_i, v_j)$ and $e_l = (v_m, v_n)$ into the dependency graph in order to capture the dependency between their output and input event model ($\delta^-_{a,out}/\delta^+_{a,out}$ and $\delta^-_{b,in}/\delta^+_{b,in}$). More precisely, $v_i = \vartheta_r(\tau_a, \delta^-_{out})$ and $v_j = \vartheta_p(\tau_b, \delta^-_{in})$ as well as $v_m = \vartheta_r(\tau_a, \delta^+_{out})$ and $v_n = \vartheta_p(\tau_b, \delta^+_{in})$. The third step then deals with the dependencies on each resource. It adds dependency edges according to the construction of the busy window (w^+/w^-), and the computation of response times (R^+/R^-). This implies that, for each scheduler, a specific transformation is necessary. Consequently, the third step must be carried out for each resource individually, respecting its scheduling analysis. Dependent tasks on a resource can either be treated as in step two following the generalized CPA theory, or be treated through the local resource analysis step, as e.g. done in [25] who considers task chains under static-priority preemptive (SPP) scheduling. The fourth step then deals with capturing the dependencies that influence the computation of the output event model, based on the resource-analysis results and the applied propagation strategy to bound them. W.l.g. we assume busy-window propagation as described by Theorems 1–3 in [29].

Dependencies are consequently expressed as edges between timing model parameters in the TDG. The TDG allows identifying timing dependencies that data which is transmitted over the network experiences.

Since the functional model $\mathcal{FG}$ has a correspondence rule with the safety viewpoint (cf. Figure 2), we can trace safety requirements from there over $\mathcal{FG}$ to individual task chains and thus to the timing model and the TDG. [16] treats safety requirements on timing requirements as so called *confidence requirements*. The confidence requirement expresses how well all timing parameters to compute a timing bound must be known, in order to utilize the computed bound as proof that the timing requirement and consequently the safety requirement is fulfilled.

The input description of components on the other hand supplies information how accurate timing parameters like WCET/BCET, e.g. payload sizes, are known. By propagating confidence values through the TDG of the system in a flow like manner where the lowest possible confidence is assigned to a TDG node, also every timing requirement in the TDG receives a confidence value. In cases where a mismatch between the assigned confidence and the confidence requirement exists, the MCC either must reject such a configuration or instantiate enforcement mechanisms to guarantee the expected model behaviour at run-time.

4.3 Monitoring and Enforcement

If the timely transmission of this data is safety relevant and dependent on parameters with lower confidence than its requirement, the MCC must take actions to bound these dependencies.

Several authors, e.g. [17, 9, 18] have proposed monitoring and enforcement techniques to conform run-time inputs to model behavior. These techniques can also be used to shape the injected traffic into the network. The MCC can deploy such mechanisms into the execution domain (cf. Figure 6). They render a dependency innocuous since they increase the confidence into a parameter to the confidence of the enforcement mechanism, which is typically high or the highest in the system. This is due to the fact that the monitors are reliable middleware components. In order to prevent overly excessive monitoring and enforcement the MCC coordinates the model enforcement strategy. Two possible strategies for efficient placement of monitors that perform enforcement are described by [16], a greedy input monitor placement and a min-cut strategy. For the MCC the greedy input placement is more suitable as it avoids complex network management where monitors would have to be implemented in the switches of the Ethernet network.

Through this enforcement, the network is guaranteed to operate within the bounds of the timing analysis. A reevaluation of the confidence values after placing enforcing monitors shows that confidence requirements are now fulfilled. Together with the timing analysis this is a sufficient proof of freedom from timing interference ([10, clause 3.75,part1]). In the case study depicted in Figure 7 the MCC performs this for the data that is transferred over the shared Ethernet network, i.e. between the Sensor Data Preprocessing and the Static Environment Modelling components, as well as for the reference trajectory sent to the vehicle control component which interfaces with the legacy control subsystem of the vehicle.

5 Conclusion

In this paper we have presented a design and integration flow that respects safety aspects of SAE level 3+ vehicle functions. We argued that the system emergent property of safety requires traceability in a design. To ensure this traceability during integration, we presented the MCC based integration flow in section 4, where traceability is inherent due to the automated model-based integration flow. This is mainly achievable due to the cross-layer model as an implementation of multiviewpoint modelling and the dependency analysis that is performed based on the cross-layer model. In section 4 we have particularly shown how this is handled for complex timing dependencies. However, the derivation and formulation of functional safety requirements for the MCC are still manual. It is our vision, to further automize the coupling between behavioral and functional safety (cf. Figure 2), i.e. integrating this aspecet in future versions of the MCC, as it is currently a manual process.

References

1 Genode OS Framework release notes 16.11, 17.08, 18.02 and 18.08. URL: `https://genode.org/documentation/release-notes/index`.

2 MAST: Modeling and Analysis Suite for Real-Time Applications. URL: `http://mast.unican.es/`.

3 A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

4 G. Bagschik, M. Nolte, S. Ernst, and M. Maurer. A System's Perspective Towards an Architecture Framework for Safe Automated Vehicles. In *2018 IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 2438–2445, 2018.

5 Peter Johannes Bergmiller. *Towards functional safety in drive-by-wire vehicles*. Springer, 2015.

6 A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. `doi:10.1109/2.30720`.

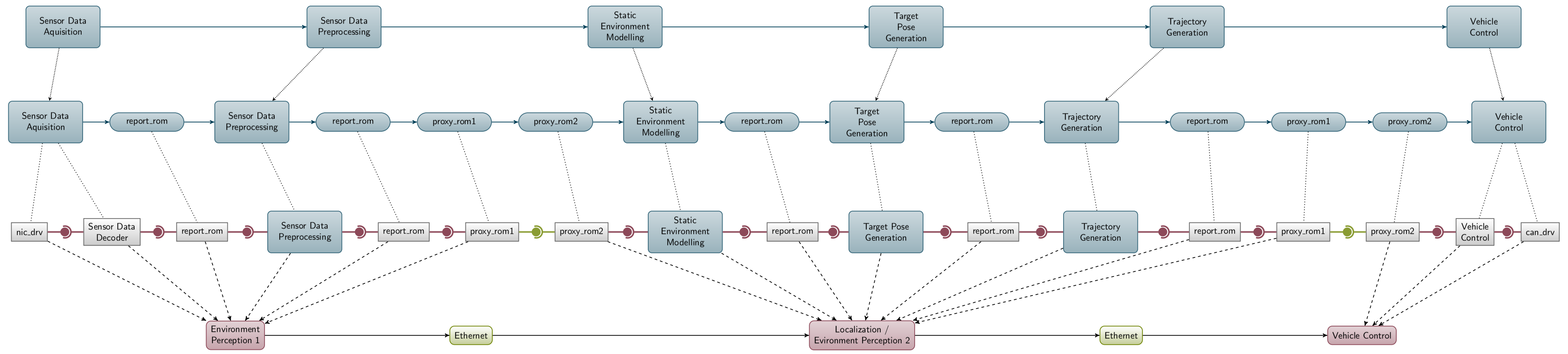


Figure 7 Visualization of how the obstacle avoidance function graph from Figure 5 is mapped to a component graph (upper half). The component graph is transformed into the component instance graph which also contains the mapping to the hardware resources (lower half). Red interface symbols indicate Genode interfaces, while green represent a data transfer over Ethernet.

7 Norman Feske. Genode OS Framework Foundations 18.05, May 2018. URL: `http://genode.org/documentation/genode-foundations-18-05.pdf`.

8 Patrik Feth, Rasmus Adler, Takeshi Fukuda, Tasuku Ishigooka, Satoshi Otsuka, Daniel Schneider, Denis Uecker, and Kentaro Yoshimura. Multi-aspect Safety Engineering for Highly Automated Driving: Looking Beyond Functional Safety and Established Standards and Methodologies. In Barbara Gallina, Amund Skavhaug, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security. SAFECOMP 2018. Lecture Notes in Computer Science*, volume 11093, pages 59–72. Springer International Publishing, 2018.

9 Kai Huang, Gang Chen, C. Buckl, and A. Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 430–436, June 2012.

10 Intern. Organization for Standardization - ISO. *ISO 26262 - Road vehicles - Functional safety*, 2 edition, April 2011.

11 Intern. Organization for Standardization - ISO. *ISO/IEC 42010 - Systems and software engineering – Architecture description*, 2011.

12 International Standard Office. *ISO/PRF PAS 21448: Road vehicles : Safety of the intended functionality*. ISO, 2018.

13 Helmut Martin, Kurt Tschabuschnig, Olof Bridal, and Daniel Watzenig. Functional Safety of Automated Driving Systems: Does ISO 26262 Meet the Challenges? In *Computer Safety, Reliability, and Security. SAFECOMP 2017. Lecture Notes in Computer Science*, pages 387–416. Springer, Cham, 2017.

14 Richard Matthaei, Gerrit Bagschik, Jens Rieken, and Markus Maurer. Stationary Urban Environment Modeling Using Multi-Layer-Grids. In *17th International Conference on Information Fusion*, 2014.

15 Mischa Möstl and Rolf Ernst. Cross-Layer Dependency Analysis with Timing Dependence Graphs. In *Proceedings of the 55th Design Automation Conference (DAC)*, 2018.

16 Mischa Möstl, Johannes Schlatow, and Rolf Ernst. Synthesis of Monitors for Networked Systems With Heterogeneous Safety Requirements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2824–2834, November 2018. `doi:10.1109/TCAD.2018.2862458`.

17 M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring Arbitrary Activation Patterns in Real-Time Systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 293–302, 2012. `doi:10.1109/RTSS.2012.80`.

18 Moritz Neukirchner, Philip Axer, Tobias Michaels, and Rolf Ernst. Monitoring of Workload Arrival Functions for Mixed-Criticality Systems. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 88–96. IEEE, 2013. `doi:10.1109/RTSS.2013.17`.

19 M. Nolte, M. Rose, T. Stolte, and M. Maurer. Model predictive control based trajectory generation for autonomous vehicles — an architectural approach. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 798–805, 2017.

20 Marcus Nolte, Gerrit Bagschik, Inga Jatzkowski, Torben Stolte, Andreas Reschka, and Markus Maurer. Towards a Skill- And Ability-Based Development Process for Self-Aware Automated Road Vehicles. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, 2017.

21 Andreas Reschka, Gerrit Bagschik, Simon Ulbrich, Marcus Nolte, and Markus Maurer. Ability and Skill Graphs for System Modeling, Online Monitoring, and Decision Support for Vehicle Guidance Systems. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 933–939, Seoul, South Korea, 2015.

22 Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. Institut für Datentechnik, 2005. URL: `http://www.digibib.tu-bs.de/?docid=00001765`.

23 J. Rieken, R. Matthaei, and M. Maurer. Toward Perception-Driven Urban Environment Modeling for Automated Road Vehicles. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 731–738, 2015.

24 SAE. J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. *Vehicle Electrification Subscription*, 2014.
25 Johannes Schlatow and Rolf Ernst. Response-Time Analysis for Task Chains in Communicating Threads. In *Real-Time Embedded Technology & Applications Symposium (RTAS)*, Vienna, Austria, April 2016.
26 Johannes Schlatow and Rolf Ernst. Response-Time Analysis for Task Chains with Complex Precedence and Blocking Relations. *International Conference on Embedded Software (EMSOFT), ACM Transactions on Embedded Computing Systems ESWEEK Special Issue*, 16(5s):172:1–172:19, September 2017.
27 Johannes Schlatow, Mischa M"ostl, and Rolf Ernst. An extensible autonomous reconfiguration framework for complex component-based embedded systems. In *12th International Conference on Autonomic Computing (ICAC)*, pages 239–242, Grenoble, France, July 2015.
28 Johannes Schlatow, Marcus Nolte, Mischa Möstl, Inga Jatzkowski, Rolf Ernst, and Markus Maurer. Towards model-based integration of component-based automotive software systems. In *Annual Conference of the IEEE Industrial Electronics Society (IECON17)*, Beijing, China, October 2017. `doi:10.24355/dbbs.084-201803221525`.
29 Simon Schliecker, Jonas Rox, Matthias Ivers, and Rolf Ernst. Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, CODES+ISSS '08, pages 185–190, New York, NY, USA, 2008. ACM.
30 T. Solte, Tianyu Liao, Matthias Nee, Marcus Nolte, and Markus Maurer. Investigating Cross-Domain Redundancies in the Context of Vehicle Automation - A Trajectory Tracking Perspective. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 2398–2405, 2018.
31 Daniel Thiele, Philip Axer, and Rolf Ernst. Improving Formal Timing Analysis of Switched Ethernet by Exploiting FIFO Scheduling. In *52nd Annual Design Automation Conference*, DAC '15, pages 41:1–41:6, New York, NY, USA, 2015. ACM.
32 Waymo. Waymo Safety Report: On the Road to Fully Self-Driving, 2017.
33 J. M. Wille, F. Saust, and M. Maurer. Stadtpilot: Driving autonomously on Braunschweig's inner ring road. In *2010 IEEE Intelligent Vehicles Symposium*, pages 506–511, June 2010.