

Model-Based Development of an Adaptive Vehicle Stability Control System*

Rasmus Adler¹, Ina Schaefer² and Tobias Schuele³

¹ Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
rasmus.adler@iese.fraunhofer.de

² Software Technology Group, Dept. of Computer Science, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
ina.schaefer@informatik.uni-kl.de

³ Embedded Systems Group, Dept. of Computer Science, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
tobias.schuele@informatik.uni-kl.de

Abstract: Safety and availability are major requirements for embedded systems in modern vehicles. However, in the automotive domain, neither conventional shutdown mechanisms nor full-fledged redundancy are appropriate means for handling faults such as failures of sensors and actuators. A suitable means for achieving the high requirements on safety and availability at low costs is graceful degradation by adapting the functionality of a system to the current driving situation and the available resources. However, adaptation significantly complicates the development of embedded systems. In this paper, we present an approach to the model-based design of adaptive embedded systems that allows coping with the increased complexity posed by adaptation. Furthermore, we show how the obtained models can be formally verified and demonstrate the feasibility of our approach by applying it to a vehicle stability control system.

Keywords: Adaptive systems, model-based development, formal verification

1 Introduction

Many advanced vehicle functions, such as antilock braking and vehicle stability control, influence elementary functionalities like braking and steering. Consequently, safety and availability are major requirements for such software-based functions. To fulfill these requirements, compensation for typical faults and deficiencies, e.g., inaccurate sensor values due to special driving situations, is indispensable. Conventional shutdown mechanisms are not appropriate for that purpose, as one software component typically contributes to many fail-operational vehicle functions. Full-fledged redundancy based on redundant hardware

*This work has been supported by the Rheinland-Pfalz Cluster of Excellence ‘Dependable Adaptive Systems and Mathematical Modelling’ (DASMOD).

is usually too expensive for automotive systems as mass products. Moreover, full-fledged redundancy might not be sufficient for compensating for conceptual faults. For instance, a homogeneous redundant sensor would deliver the same inaccurate sensor values and could not remedy any fundamental deficiencies.

To solve these problems, dynamic adaptation of system components and graceful degradation of the system's functionality have become the state of the art in automotive systems for reacting to dynamically changing driving situations as well as to failures. As many functions in an automobile are based on approximated physical models assuming certain driving situations, it is necessary to adapt to the most reasonable calculation variant for a sensor value depending on the current situation. Additionally, a system may downgrade its functionality autonomously if a failure cannot be compensated by dynamic adaptation, which is referred to as graceful degradation [She03]. For example, if the sensor measuring the yaw rate of a car fails, the vehicle stability control system may adapt to a configuration where the yaw rate is approximated by steering angle and vehicle speed. Of course, the systems may upgrade again in case of transient failures to provide the best functionality possible with the currently available resources.

However, adaptation significantly complicates the development of embedded systems due to the fact that adaptation of a component affects the quality of the services it provides, which may in turn cause adaptations in other components. As a result, sequences of adaptations may take place that are hard to analyze. For this reason, it is not sufficient to consider each configuration separately to ensure system correctness. Instead, the adaptation process as a whole has to be checked, which is a complex and error-prone task, at least without tools for computer-aided verification, since the number of configurations a system may adapt to is, in the worst case, exponential in the number of components.

A promising approach to deal with the increased complexity posed by adaptation is model-based design. In this paper, we present a constructive modeling technique for the design of adaptive embedded systems. As a major advantage, our approach hides the complexity at the system level by fostering independent specification of functionality and adaptation behavior. To this end, the adaptation behavior of the components is specified independently from the functionality. The adaptation behavior of the whole system is determined by the adaptation behavior of the system's components. Furthermore, our approach allows the designer to analyze, validate, and to verify the adaptation behavior. The integration of formal verification into the development process is particularly important to prove that the adaptation behavior meets critical requirements such as deadlock-freeness. We demonstrate the feasibility of our approach by applying it to a vehicle stability control system, which supports dynamic adaptation to react to changing driving situations and graceful degradation in case of failures.

The remainder of this paper is structured as follows: In Section 2, we present the vehicle stability control system. Moreover, we show how adaptation and graceful degradation can be used to increase safety and availability. Section 3 discusses our modeling concepts for the development of adaptive embedded systems. In Section 4, we propose a development process integrating modeling and analysis. Experimental results on the verification of the vehicle stability control system are given in Section 5. Finally, we discuss related work in Section 6 and conclude with a summary and an outlook to future work in Section 7.

2 Vehicle Stability Control System

Our vehicle stability control system consists of three typical software-intensive vehicle functions, which have been implemented on a remote controlled car scaled 1:5. The first vehicle function is a steering angle delimiter, which restricts the steering angle depending on the current velocity of the car. This function helps to prevent the car from spinning if the steering angle intended by the driver might cause instabilities. The second function is a traction control system, which controls the slip of the rear wheels. If the slip exceeds a given threshold, the acceleration of the car is restricted by reducing the gas to prevent loss of traction at drive-away. The third function implements a yaw rate corrector influencing the brakes. For example, if the driver wants to go straight, the system tries to keep the yaw rate close to zero. Normally, a yaw rate controller can be used in any driving situation. In our system, however, the function only considers straightforward driving. This simplification is well-suited for our purposes, since it facilitates the implementation of the functional behavior, while providing the same challenges for modeling and analyzing adaptation and graceful degradation.

In order to implement these vehicle functions, the car is equipped with typical sensors: At each wheel, a sensor is mounted for measuring the revolution. Additionally, there are sensors measuring longitudinal acceleration, lateral acceleration, and yaw rate. Besides the values measured by the sensors, the system receives as input the desired speed and the desired steering angle via remote control from the user. The system controls four servos: The first servo controls the gas and actuates the rear brakes. Hence, the rear wheels are decelerated with the same force. The front brakes have their own servos and are actuated independently from each other. The fourth servo controls the steering.

The system architecture is split into logical sensors, controllers, and logical actuators as depicted in Figure 1. The logical sensors/logical actuators implement the mapping between physical sensor/actuator values and their logical interpretation (e.g., slip of wheels and nominal yaw rate). The controller component contains three subcomponents implementing the vehicle functions described above based on logical sensor and actuator values.

The components in *LogicalSensors* are self-adaptive, since the most reasonable calculation variant for a sensor value typically depends on the current driving situation. This even holds for logical sensor values, where the relation to measured sensor values seems to be straightforward. For example, component *AyCalc*, which determines the lateral acceleration ay , implements two different modes of operation, each providing a different quality level. In the following, we will call these externally ‘visible’ modes of operation *configurations*. For instance, configuration *Measured* derives ay from the values provided by an acceleration sensor. However, when the car is going down- or uphill, the acceleration sensor values are influenced by gravity. For this reason, an alternative configuration *VYawVCarRefBased* is used, which calculates ay from the speed of the car along the x-axis (v_{carref}) and the angular speed of the car around its z-axis (v_{yaw}).

Dynamic adaptation in the logical sensor components is a powerful and cost-efficient way to minimize the effects of sensor defects. In many cases, it is even possible to fully compensate the occurrence of faults. For instance, the component *VYawCalc* has four different

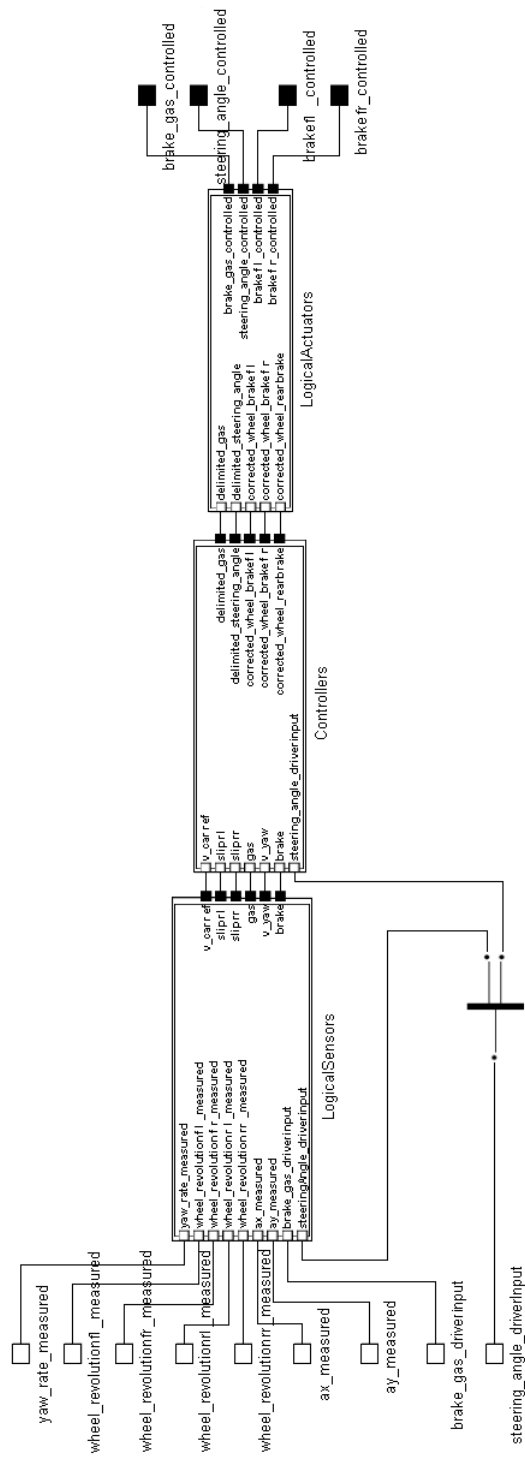


Figure 1: Architecture of the vehicle stability control system

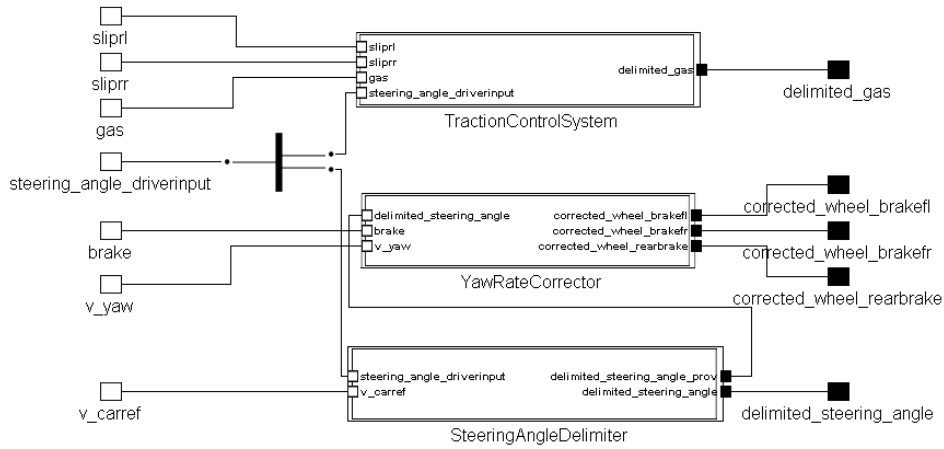


Figure 2: Internal structure of the controller component

configurations for determining the angular speed of the car around its z-axis (v_yaw). Configuration *Measured* uses a yaw rate sensor, configuration *SteeringBased* derives v_yaw from the steering angle and the speed of the car along the x-axis (v_carref), configuration *RWheel* uses the rear wheel speeds to determine v_yaw , and configuration *AyBased* uses the lateral acceleration a_y and v_carref . In all configurations, the quality of the provided value is sufficient to perform a yaw rate correction. This means that component *VYawCalc* is able to adapt to the most appropriate configuration according to the availability of the sensors without downgrading the functionality of the system.

In contrast to the logical sensors and actuators, the configurations of the controller components directly correspond to degradation levels of the vehicle functions. Since the considered vehicle functions are reasonably simple, each of the three controller components shown in Figure 2 implements only one safe fall-back layer. Besides the configuration that provides the full functionality, there is another configuration that implements a safe subset of the functionality based on minimal input. The idea behind this fall-back configuration is to ensure that the system is still operational even if the full functionality cannot be provided. Additionally, each component has a configuration *Off* in which the component is shut down. This configuration is primarily used for validation and verification, as shutting down whole components is undesirable in practice.

As an example of a controller component, consider *TractionControlSystem*, which has the configurations *RearWheelSlipControl*, *SlowStart*, and *Off*. The configuration *RearWheelSlipControl* only increases the gas if the slip is below a given threshold. This configuration requires the slip of the right and the left rear wheels as well as the intended gas value for calculating the delimited gas value. In order to avoid a total loss of the delimited gas value when the slip of the wheels cannot be determined or is erroneous for some reason, the component has the fall-back configuration *SlowStart*, which only requires the desired gas value as input. This configuration limits the increase of the gas in a very conservative way by taking into account the last desired gas value.

3 Modeling Adaptation Behavior

As mentioned previously, one major problem in specifying how a system has to adapt in case of failures stems from complex interdependencies between different components. These interdependencies are due to the fact that adaptation of a component may affect the quality of the service it provides, which has to be taken into account by the influenced components. Hence, adaptation of a component may trigger a chain reaction of adaptations in other components until a valid system configuration is reached. As another problem, most vehicle functions are not disjoint, i.e., a component may be used by more than one function at a time.

For these reasons, a constructive modeling technique hiding the complexity at the system level is essential for specifying complex adaptation behavior. In our research project MARS (Methodologies and Architectures for Runtime Adaptive Embedded Systems), we have developed a modeling approach that separates functionality from adaptation behavior by establishing a quality flow in the system [TAFJ07]. This allows encapsulating the specification of the adaptation behavior and modeling it as a separated view [MBE⁺00] within the components. In this way, developers can focus on the adaptation behavior of single components until the whole system is specified. In a second step, validation and verification techniques may be employed to check correctness of the adaptation behavior at the system level.

3.1 Quality Flow

In architecture description languages [MT00], component interfaces are defined by input and output ports. Each port has an associated data type and can be linked with other ports. The relationship between the data types is often formalized in a global type system, which makes it possible to perform automatic type checks. Components communicate with each other using signals, which are, from a functional point of view, instances of data types.

Corresponding to interfaces in architecture description languages and the data flow between components, we introduce adaptation interfaces and establish a quality flow based on instances of quality types (QT). QTs are used to augment data values with a description of their quality. All QTs are part of a quality type system (QTS), which is an inheritance tree with root node *BASICQT* (see Figure 3).

The root QT (*BASICQT*) has two subtypes *AVAILABLE* and *UNAVAILABLE*. The latter indicates that a data value is unusable due to defects, whereas the former indicates that a data value has no defect making it unusable. All other QTs in the QTS are inherited from *AVAILABLE* and have semantics describing the intuition behind the related data value. For instance, the QT *MEASURED* referring to data values calculated by configuration *Measured* of component *AyCalc* has the meaning ‘gravity-influenced lateral acceleration based on acceleration measurements’. The configuration *VYawBased* of the component *AyCalc* delivers values with the semantics ‘slip-influenced lateral acceleration based on calculations using *v_carref* and *v_yaw*’ (QT *VYAW_BASED*). The QT *AY* with semantics ‘lateral

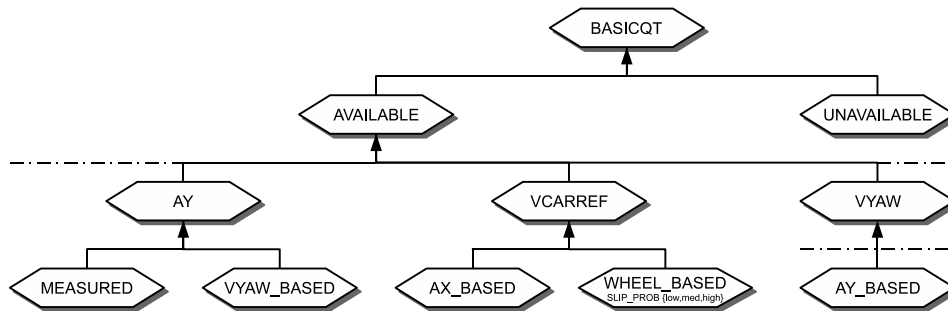


Figure 3: Example for a quality type system (QTS)

acceleration’ subsumes these two different QTs for lateral acceleration. Moreover, a QT may define a set of attributes enabling a quantitative and more precise description of the quality. For example, the QT *WHEEL_BASED*, which means ‘wheel-based approximation of car speed in x-direction’ has an attribute *SLIP_PROB* with the range $\{low, med, high\}$ providing information about the likelihood that a calculated value is affected by the slip of the wheels.

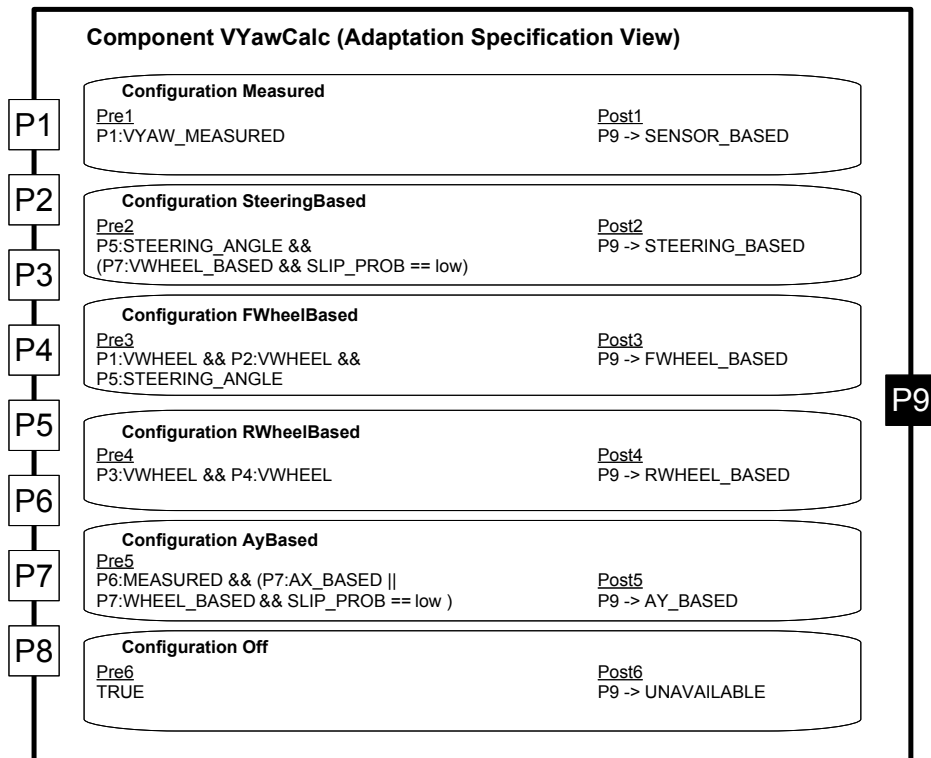
3.2 Specification of Adaptation Behavior

In analogy to the functional interface, the adaptation interface is defined by quality ports. For every data port, a quality port is added to the adaptation interface. Quality ports have a reference to a QT and are declared either as *required* or *provided*. In the same way as the distinction between input and output ports defines the direction of the data flow, the distinction between *required* and *provided* ports defines the direction of the quality flow. Connectivity checks between ports can be extended to the adaptation interface and inconsistent adaptation behavior through directly connected components can be identified automatically. Quality type checks are performed by verifying that the QT of a *provided* port is equal to or inherited from the QT of the connected *required* port.

To specify the adaptation behavior, we attach to each configuration a pair consisting of a precondition and a postcondition. The precondition is used to select the current configuration according to the QTs at the *required* ports of the adaptation interface. The postcondition assigns a QT to every *provided* port of the adaptation interface. The preconditions are evaluated at runtime, where the one with the highest priority, which is implicitly given by their order, determines the configuration to be selected. Once the selected configuration has become active, the QTs at the *provided* ports are set according to the given postcondition.

For example, the component *VYawCalc* contains nine quality ports and six configurations (see Figure 4). The eight quality ports referring to input ports are declared as *required* and the quality port referring to the output port *v_yaw* is declared as *provided*. The configuration *AyBased* implements a yaw rate calculation from the measured velocity *v_carref* and

the lateral acceleration ay . Its precondition states that the quality at port ay is an instance of QT *MEASURED*. A quality of QT *VYAW_BASED* is forbidden, since it is not reasonable to derive v_yaw from ay and ay from v_yaw at the same time. The quality at the port v_carref has to be an instance of QT *AX_BASED* or *WHEEL_BASED*. If the quality is of QT *WHEEL_BASED*, then the attribute *SLIP_PROB* must have the value *low*. The last pair of conditions always refers to the configuration *Off*. Clearly, the corresponding precondition has always to be valid and the postcondition assigns the QT *UNAVAILABLE* to all provided qualities.



port number	port name	referenced QT	quality flow
P1:	yawrate	VYAW_MEASURED	required
P2:	v_wheelfl	VWHEEL	required
P3:	v_wheelfr	VWHEEL	required
P4:	v_wheelrl	VWHEEL	required
P5:	v_wheelrr	VWHEEL	required
P6:	steering_angle	STEERING_ANGLE	required
P7:	ay	AY	required
P8:	v_carref	VCARREF	required
P9:	v_yaw	VYAW	provided

Figure 4: Adaptation specification for component *VYawCalc*

4 Development Process for Adaptive Systems

In the following, we will briefly describe the design flow of our approach, which is depicted in Figure 5. This design flow has been used successfully for the development of the vehicle stability control system.

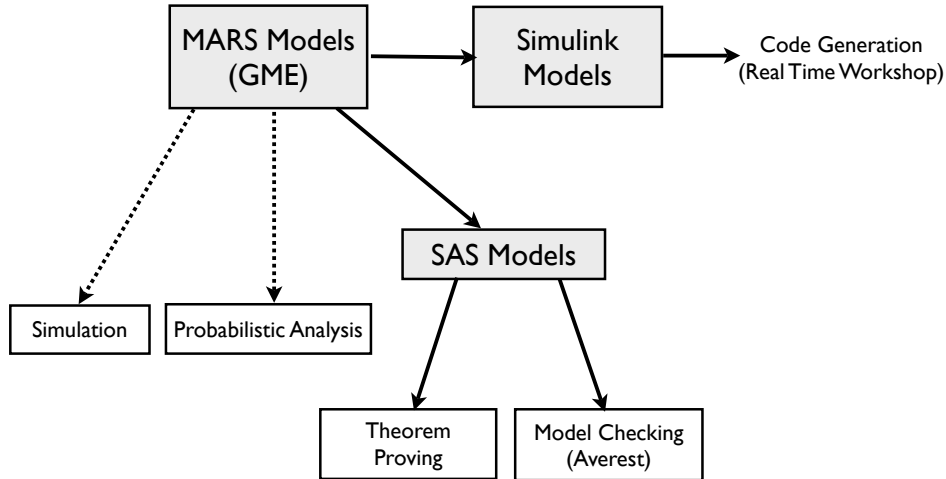


Figure 5: Design flow for the development of adaptive systems

The MARS modeling concepts have been integrated into the Generic Modeling Environment GME¹ [LMB⁺01] by providing a GME meta-model. GME automatically produces a model representation in XML format, which can be used as input for validation and verification as well as for code generation. Regarding validation, we support the simulation of adaptation behavior and the visualization of reconfiguration sequences using adaptation sequence charts [TAFJ07]. Moreover, it is possible to perform a probabilistic analysis of the adaptation behavior [AFT07]. To this end, the adaptation behavior model is translated into an equivalent hybrid component fault tree. The probability that a configuration of a component is activated can then be derived from the failure rates of the sensors and actuators.

Moreover, MARS models can be formally verified by automatically translating them into synchronous adaptive systems (SAS) [ASSV07]. SAS are a formal framework that allows capturing MARS models at a high level of abstraction and reasoning about their semantics by theorem proving techniques. Moreover, SAS can be used to perform model reductions, e.g., slicing [SPH08], in order to make the models amenable to automatic verification tools, e.g., symbolic model checkers. The correctness of the reductions is established with a translation validation technique [BSPH07] by generating proof scripts for an interactive theorem prover [NPW02].

¹<http://www.isis.vanderbilt.edu/projects/gme/>

As a model checking back-end, we use Averest,² a framework for the specification, implementation, and verification of reactive systems [SS05]. In Averest, a system is given in a synchronous programming language, which is well-suited for describing adaptive systems obtained from SAS models, as both are based on a synchronous semantics. Furthermore, a causality analysis of synchronous programs can be used to detect cyclic dependencies that may occur if the quality flow generated by an output is used as input in the same component. Specifications can be given in temporal logics such as CTL and LTL (see Section 5).

Once the adaptation behavior of a MARS model has been successfully validated and verified, it can be implemented by means of a given adaptation framework. Such a framework might either be a central component in the system coordinating all adaptation processes or be distributed over several components. We have developed a distributed framework in MATLAB-Simulink,³ which provides an interface for the integration of the functionality and supports code generation using Simulink’s Real-Time Workshop. Additionally, the framework offers a generic configuration transition management, which can be used to specify the interaction between functionality and adaptation behavior in more detail. The resulting design can then be evaluated by co-simulations of functional and adaptation behavior. In order to avoid undesirable interference between functional and adaptation behavior, additional techniques may be employed that guarantee smooth blending between functional outputs during reconfigurations [Bei07].

5 Experimental Results

In this section, we present experimental results on the verification of the vehicle stability control system using model checking. Table 1 shows some characteristics of the system.

Number of components	28
Number of configurations	70
Lines of code	≈ 2500
Number of reachable states	$\approx 5 \cdot 10^{18}$
Number of properties	151

Table 1: Characteristics of the vehicle stability control system

As specification languages for reasoning about adaptation behavior, we employ the temporal logics CTL (computation tree logic) and LTL (linear time temporal logic) [CGP99, Sch03]. In both CTL and LTL, temporal operators are used to specify properties along a computation path that corresponds to an execution of the system. For example, the formula $F\varphi$ states that φ eventually holds and $G\psi$ states that ψ invariantly holds on a given path. In CTL, every temporal operator must be immediately preceded by one of the path quantifiers A (for all paths) and E (at least one path). Thus, $AG\varphi$ and $EF\psi$ are CTL formu-

²<http://www.averest.org>

³<http://www.mathworks.com>

lae stating that φ invariantly holds on all paths and ψ eventually holds on at least one path, respectively. LTL formulae always have the form $A\varphi$, where φ does not contain any path quantifiers. None of these two logics is superior to the other, i.e., there are specifications that can be expressed in LTL, but not in CTL, and vice versa [CGP99, Sch03].

Most of the properties concerning adaptation behavior can be divided into four generic classes of CTL formulae. First, we want to verify that no component gets stuck in the shutdown configuration *Off*. The following specification states that whenever a component is in configuration *Off*, there exists a path such that the component is eventually in a configuration different from *Off* (the variable c stores the current configuration).

Property 1 (liveness): $AG(c = Off \rightarrow EF c \neq Off)$

The next property states that every component can reach all configurations at all times. If this specification holds, the system is deadlock-free and no configuration is redundant.

Property 2 (reachability): $AG(\bigwedge_{i=1}^n EF c = config_i)$

Moreover, a component must always be in one of the predefined configurations such that no inconsistent states can be reached.

Property 3 (safety): $AG(\bigvee_{i=1}^n c = config_i)$

Additionally, we want to verify that no configuration is always only transient, i.e., that it can be active for an arbitrary amount of time. The following formula holds iff a component can stay in a configuration.

Property 4 (persistence): $\bigwedge_{i=1}^n EFEG c = config_i$

Finally, an important property of adaptive systems is stability [SPH06, ASSV07]. Since adaptation in our approach is not controlled by a central authority, adaptation in one component may trigger further adaptations in other components. While *finite* sequences of adaptations are usually intended, cyclic dependencies between components may lead to an *infinite* number of adaptations, which results in an unstable system. For this reason, we want to verify that the configurations of a component eventually stabilize if the inputs do not change. In contrast to Properties 1–4, stability cannot be expressed in CTL, but in LTL. Suppose that φ_{input} holds iff the inputs are stable for one unit of time. Moreover, let φ_{config} hold iff the configurations are stable for one time unit. Then, the system is stable iff the following formula holds:

Property 5 (stability): $AG(G\varphi_{input} \rightarrow FG\varphi_{config})$

Intuitively, the formula states that the configurations will stabilize after a finite amount of time whenever the inputs do not change. In principle, we could use standard model checking procedures for LTL to verify stability. However, as shown in [ASSV07], this is often rather inefficient. For this reason, we employed the approach presented in [ASSV07], which turned out to be more efficient in practice (see below).

Table 2 shows the number of fixpoint iterations performed during model checking and the time required to check Properties 1–4 on a Pentium 4 processor with 2.8 GHz and 512 MB RAM. As these properties actually represent sets of formulae (one formula for each component/configuration), we have determined minimal, average, and maximal values in addition to the total ones. The total time required to check all properties was less than three minutes. Stability could not be checked in less than one hour by means of standard model checking procedures for LTL. However, using the approach presented in [ASSV07], it could be checked in less than half an hour (1723 seconds). Nevertheless, there is still a large gap compared to the runtimes for Properties 1–4. The main reason for this is that the latter can benefit from abstractions such as cone of influence reduction [CGP99], whereas checking stability requires considering the complete system.

Property	Fixpoint Iterations				Time [seconds]			
	Min.	Avg.	Max.	Total	Min.	Avg.	Max.	Total
P1 (liveness)	5	8.3	17	223	< 0.1	3.1	71.5	84.0
P2 (reachability)	9	18.4	43	496	< 0.1	2.4	52.1	63.8
P3 (safety)	2	2.0	2	54	< 0.1	0.1	0.2	0.7
P4 (persistence)	6	10.8	20	758	< 0.1	0.3	4.5	19.6

Table 2: Experimental results of model checking

It should be mentioned that not all of these properties were satisfied in the initial design of the vehicle stability control system. For example, it turned out that the configuration *FWheelBased* of the component *VYawCalc* was never active for more than a single cycle (violation of Property 4). This behavior was due to interdependencies between the component *VYawCalc* and other components that caused it to switch to the configuration *SteeringBased* in the next cycle. Although this particular case might not be safety-critical, the results of verification are often useful for gaining a better understanding of the whole system.

Besides the properties described above, which are concerned with generic features of the adaptation behavior, we also checked a number properties ensuring that the system always provides its basic functionality given that the minimal required sensors and actuators are operational. In particular, if the sensors and actuators concerning steering angle, brake force, and gas are operational, neither of the corresponding controller components is in the shutdown configuration *Off*. In this way, it is guaranteed that the adaptation behavior implemented in the system does not corrupt any essential functionality. For example, if the intended steering angle is available and the steering servo works, i.e., *delimited_steering_angle_quality* is available, then the component *SteeringAngleDelimiter* is not in the configuration *Off*:

$$AG((steering_angle_driverInput_quality = available \wedge delimited_steering_angle_quality = available) \rightarrow c(SteeringAngleDelimiter) \neq Off)$$

As another example, the following property ensures that the braking system does not fail

if the required actuators are operational:

$$\text{AG}((\text{corrected_wheel_brakeFL_quality} = \text{available} \wedge \\ \text{corrected_wheel_brakeFR_quality} = \text{available} \wedge \\ \text{corrected_wheel_rearBrake_quality} = \text{available}) \rightarrow c(\text{YawRateCorrector}) \neq \text{Off})$$

6 Related Work

In recent years, a number of frameworks for dynamic adaptation and reconfiguration have been developed, e.g., [RL05, COWL02]. Most of these frameworks are based on a dedicated control that contains all information on the adaptation behavior and triggers adaptation of the components. Thus, even minor changes in one of the components might require a complete redesign of the component controlling the adaptation behavior. In contrast, our approach does not depend on a central control encoding the complete adaptation behavior. This simplifies the design of adaptive systems and improves their maintainability.

A major challenge in the model-based development of adaptive systems is to find models that lead to reasonable and safe degradation behavior. To solve this problem, the verification of adaptation behavior has become an active area of research [KB04, Str05, ZC05, ZC06]. However, these approaches take the specification of the system for granted and provide no constructive modeling technique. Moreover, directly specifying the global adaptation behavior as required by these approaches is hardly feasible for complex systems like the Electronic Stability Program (ESP).

Regarding the modeling of adaptation behavior, approaches like [SKN01] or the MUSIC project⁴ have adopted ideas from variability modeling used to manage product lines. The basic idea of these approaches is to shift the binding time of variation points from design time to runtime. Thus, the systems autonomously determine all valid configurations at runtime and select one of them according to an objective function. However, determining the next configuration in this way is computationally intensive and thus too slow for real-time systems. Furthermore, the complexity of computing the next configuration strongly limits model-based analysis, validation, and verification, which are indispensable in safety-critical areas like the automotive domain.

For a detailed description of MARS, the reader is referred to [TAFJ07]. Further information on the translation of MARS models to SAS and their verification can be found in [ASSV07]. Initial work on the verification of MARS models was presented in [SST06]. In this paper, we improved on [TAFJ07, ASSV07] in two ways: First, we introduced a new quality type system based on inheritance trees. Second, we applied our approach to a vehicle stability control system to demonstrate its feasibility for large real-world examples.

⁴<http://www.ist-music.eu/>

7 Conclusion and Future Work

Adaptation and graceful degradation have become the state of the art in the automotive domain to meet the high demands on safety and availability. Adaptation is frequently employed to react to changing driving situations that require switching between different modes of operation. Graceful degradation enables a system to continue operating properly in case of failures, e.g., defective sensors. However, adaptation and graceful degradation significantly complicate the design of a system.

We presented a constructive modeling approach for the development of adaptive embedded systems, which facilitates independent specification of adaptation behavior and functionality. This helps the designer to focus on each aspect separately and to resolve complex interdependencies. Furthermore, conventional techniques for the design of fault-tolerant systems can be integrated smoothly into our approach. For example, fault-tolerance with respect to certain input values can be modeled by a scenario where a component adapts to a configuration that does not affect the quality of the provided service.

Our approach supports the integration and belated specification of legacy components, e.g. components whose adaptation behavior is unspecified. This is accomplished by assigning the basic quality type *BASICQT* to every quality port and by establishing appropriate preconditions/postconditions. The component's adaptation behavior can then be successively refined until the desired behavior is obtained. In this way, ASCET-SD models from the industry have been successfully annotated.

In safety-critical areas, careful analysis of the adaptation behavior is a crucial concern, as interdependencies between the components of a system may result in complex adaptation sequences that are hard to analyze. Besides validation techniques such as simulation, our approach also supports formal verification. Our experimental results indicate that most properties concerning the adaptation behavior of complex systems such as the vehicle stability control system can be verified efficiently within a couple of minutes.

In our future work, we want to extend the approach presented in this paper by concepts for the hierarchical specification and compositional analysis of adaptive systems. Furthermore, we plan to model at a high level of abstraction the relationship between a configuration and its underlying functional behavior. In this way, the functionality can be analyzed in combination with the adaptation behavior at the modeling level before the functionality is actually implemented.

References

- [AFT07] R. Adler, M. Förster, and M. Trapp. Determining Configuration Probabilities of Safety-Critical Adaptive Systems. In *International Symposium on Ubisafe Computing (UbiSafe'07)*, Niagara Falls, Canada, 2007. IEEE Computer Society.
- [ASSV07] R. Adler, I. Schaefer, T. Schuele, and E. Vecchié. From Model-Based Design to Formal Verification of Adaptive Embedded Systems. In *International Conference on Formal*

- Engineering Methods (ICFEM'07)*, volume 4789 of *LNCS*, pages 76–95, Boca Raton, USA, 2007. Springer.
- [Bei07] Andreas Beicht. Entwicklung eines Frameworks zur Entwicklung und Analyse adaptiver eingebetteter Systeme, 2007. Diplomarbeit, TU Kaiserslautern, Germany.
- [BSPH07] J.O. Blech, I. Schaefer, and A. Poetzsch-Heffter. Translation Validation of System Abstractions. In *Workshop on Runtime Verification (RV'07)*, volume 4839 of *LNCS*, pages 139–150, Vancouver, Canada, 2007. Springer.
- [CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT, London, England, 1999.
- [COWL02] J.M. Cobleigh, L.J. Osterweil, A. Wise, and B. Lerner. Containment Units: A Hierarchically Composable Architecture for Adaptive Systems. In *Symposium on Foundations of Software Engineering (SIGSOFT FSE'02)*, Charleston, USA, 2002. ACM.
- [KB04] S.S. Kulkarni and K.N. Biyani. Correctness of Component-Based Adaptation. In *Symposium on Component Based Software Engineering (CBSE'04)*, volume 3054 of *LNCS*, pages 48–58, Edinburgh, Scotland, 2004. Springer.
- [LMB⁺01] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The Generic Modeling Environment. In *Workshop on Intelligent Signal Processing (WISP'01)*, Budapest, Hungary, 2001. IEEE.
- [MBE⁺00] S. Mann, A. Borusan, H. Ehrig, M. Grosse-Rhode, R. Mackenthun, A. Sünbül, and H. Weber. Towards a Component Concept for Continuous Software Engineering. Technical Report 55/00, Fraunhofer-ISST, 2000.
- [MT00] N. Medvidovic and R.N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1), 2000.
- [NPW02] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- [RL05] O.A. Rawashdeh and J.E. Lumpp, Jr. A Technique for Specifying Dynamically Reconfigurable Embedded Systems. In *Aerospace Conference*, Big Sky, USA, 2005. IEEE.
- [Sch03] K. Schneider. *Verification of Reactive Systems – Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer, 2003.
- [She03] C.P. Shelton. *Scalable Graceful Degradation for Distributed Embedded Systems*. PhD thesis, University of Pittsburgh, Pennsylvania, USA, 2003.
- [SKN01] C.P. Shelton, P. Koopman, and W. Nace. A Framework for Scalable Analysis and Design of System-Wide Graceful Degradation in Distributed Embedded Systems. In *Workshop on Reliability in Embedded Systems*, New Orleans, USA, 2001. IEEE.
- [SPH06] I. Schaefer and A. Poetzsch-Heffter. Towards Modular Verification of Stabilisation in Self-Adaptive Embedded Systems. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06)*, volume 4280 of *LNCS*, pages 584–585, Dallas, USA, 2006. Springer.
- [SPH08] I. Schaefer and A. Poetzsch-Heffter. Slicing for Model Reduction in Adaptive Embedded Systems Development. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008)*, 2008.

- [SS05] K. Schneider and T. Schuele. Averest: Specification, Verification, and Implementation of Reactive Systems. In *Conference on Application of Concurrency to System Design (ACSD'05)*, St. Malo, France, 2005.
- [SST06] K. Schneider, T. Schuele, and M. Trapp. Verifying the Adaptation Behavior of Embedded Systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS'06)*, pages 16–22, Shanghai, China, 2006. ACM.
- [Str05] E.A. Strunk. *Reconfiguration Assurance in Embedded System Software*. PhD thesis, University of Virginia, Charlottesville, USA, 2005.
- [TAFJ07] M. Trapp, R. Adler, M. Förster, and J. Junger. Runtime Adaptation in Safety-Critical Automotive Systems. In *IASTED International Conference on Software Engineering (SE'07)*, Innsbruck, Austria, 2007. ACTA.
- [ZC05] J. Zhang and B.H.C. Cheng. Specifying Adaptation Semantics. In *Workshop on Architecting Dependable Systems (WADS'05)*, pages 1–7, St. Louis, USA, 2005. ACM.
- [ZC06] J. Zhang and B.H.C. Cheng. Model-Based Development of Dynamically Adaptive Software. In *International Conference on Software Engineering (ICSE'06)*, pages 371–380, Shanghai, China, 2006. ACM.