EVS26
Los Angeles, California, May 6 - 9, 2012

Early Simulation and Testing of Virtual ECUs for Electric Vehicles

Lars Stockmann, Dominik Holler, Dr. Dirk Spenneberg
dSPACE GmbH, Rathenaustraße 26, 33102 Paderborn, LStockmann@dspace.de

Abstract

Testing new concepts in the field of closed-loop control is an important matter to the automotive industry. Especially in the domain of electric vehicles, there is a lot of ECU functionality targeting energy management and engine controlling that has to be developed and evaluated. Over the last decade, hardware-in-the-loop simulation has successfully been used to test new ECUs. However, in early stages of development, the target hardware is often not known and only parts of the final implementation is available. Furthermore, there are different domains involved in the development process, from an early behavioral model to the final integration into an ECU.
In our project we propose a methodology and tool chain that uses the simulation of virtual ECUs across multiple domains. Virtual ECUs enable testing the functionality together with hardware-independent non-application relevant aspects (e.g. some parts of the basic software or the run-time environment), which effectively postpones the need for a close-to-production ECU prototype.
In this paper, we compare different levels of ECU abstractions regarding their convergence to a real ECU and certain requirements that are imposed when developing electric vehicles. Here, we specifically aim at finding the best suitable level for the early stages of development, where many aspects of the final ECU are still not known and a HIL setup is not feasible. Nevertheless, one major goal of our project is to experiment and test the virtual ECU in the same tool environment that is commonly used in HIL setups to provide a certain reusability. In this context, this paper also presents current approaches for the reuse of tests across domains, which will be addressed further as our project advances.

Keywords: control system, hardware-in-the-loop (HIL), modeling, optimization, simulation

1 Introduction

Today's automotive industry faces many challenges when developing zero emission cars that use hydrogen fuel cells or battery-based solutions. Even though both fields benefit from the industry's well-established methodologies and know-how, it is especially in the area of closed-loop control where many aspects require new concepts and approaches. One example is the integration of new, highly efficient energy management solutions.
To meet the high safety requirements of the vehicle industry, all electric control units (ECUs) must undergo a long series of tests, from the early mathematical models of the functionality (e.g. a MATLAB®/Simulink® model) to the full-fledged ECU software, including the operating system and bus communications. Many of these tests are performed using closed-loop simulations that use environment models to generate the proper feedback. Furthermore, modern standards emerge that address the growing complexity of automotive software. The AUTomotive Open System ARchitecture or AUTOSAR for short provides a standardized automotive software architecture. It facilitates the exchangeability of ECU software and hardware [1].
In early phases, functional models are developed and tested using model-in-the-loop (MIL) sim-

ulation. In a next step, a software developer uses the functional model as a specification and implements the C code or an implementation model. In the latter case, an ECU autocoding tool like dSPACE TargetLink uses this implementation model to generate the C code. This application code is then tested with software-in-the-loop (SIL) or, if an evaluation board is available, with processor-in-the loop (PIL) simulation (e.g. to verify that a target compiler produces correct code). The results of this simulation are compared to the former MIL simulation. Tests using SIL/PIL simulation in this context allow to find issues that are induced by integer arithmetic (as opposed to floating-point arithmetic in a MIL simulation) like integer overflow or quantization errors. It should be noted though, that the software components are tested individually. Errors that result from interacting with other components remain undiscovered.

Until that point, the code can be tested on off-the-shelf personal computers (if the target hardware is known, an evaluation board can be used). Afterwards, the tested code is passed to the software integrator who integrates it with non-application-related code (often named basic software, e.g. operating system, diagnostic event manager, I/O, bus drivers), and produces flashable code for the real hardware-target. This basic software layer is often also referred to as hardware-dependent software [8], which can be misleading as this paper shows that not all aspects of this layer might actually depend on the final hardware.

Beyond that, in modern architectures like AUTOSAR a run-time environment is implemented as a middleware between the application layer and basic software layer. The AUTOSAR Runtime Environment (RTE) abstracts inter- and intra-ECU communication of the application software. Thus, the application software does not need to be aware of the communication specification, which can be configured independently.

The integrated ECU code together with a first version of the ECU hardware must now be tested using hardware-in-the-loop (HIL) simulation. HIL simulation has become a firmly established means for the late phase of development [22, 24].

However, such tests need expensive ECU prototypes that often are not available at an early phase of development. Moreover, basic software modules might not yet be implemented or ready for integration. Still, it would be interesting to know, how the software components interact with each other and/or with an already configured basic software module like a diagnostic event manager. Thus, it would be great benefit if the simulation of the application software in combination with available aspects of basic software would be possible even when a real ECU prototype is not at hand. This needs a model of an ECU that adequately abstracts its hardware and the missing basic software components. With such a model of an ECU it is possible to use conventional PCs as simulation a platform. A key difference to a HIL simulation is that the simulation is usually not done in real-time. This means that a whole simulation can be done in less time, only depending on the speed of the underlying PC. Also, it allows pausing the simulation at any time for debugging purposes. However, it is also the reason why such a simulation will never replace the later HIL tests, because there are hardware-dependent aspects of the ECU software that need the ECU prototype and the real-time capabilities of a HIL simulator. If anything, such a model can be seen as a link between the pure application software and the ECU prototype. If, for example, the simulation of the ECU model is sufficiently close to the real behavior and allows the use of the typical experimentation and test tools used with a real HIL simulator, it would be possible for the HIL tester to implement and execute his/her tests for the later HIL simulation beforehand. Ideally, tests (especially functional tests) could be reused on the HIL setup. This would significantly reduce the time needed for later HIL test preparation.

There are a number of approaches providing different levels of abstraction: i.e. the virtual ECU (V-ECU) that appeared in 2007 [21] and, during the last years, has become an established means for virtual simulation-based validation and verification [15]. A V-ECU emulates the behavior of the later ECU based on software that already exists in early stages (i.e. parts of the application software, and certain basic software configurations). This enables testing the ECU software on a higher integration level (compared to the individual simulation of application code) before it is implemented on a real target (front-loading of tests). Thus, errors can be found early in the development process and development costs can be reduced.

The question is whether this approach can be also used in the context of electric vehicles, where new control concepts are developed from the start and time-to-market pressure is high. Also, compared to fuel-based cars, the whole topology of the electric control units may have to be realized differently due to the special need for weight reduction and a more compact design. These are only two use cases typical for developing electric vehicles. Of course many other use cases are inherited from conventional automotive development. Therefore we want to investigate:

- which level of abstraction is actually required in the early development stages of electric vehicles
- how such a model of an ECU can be used in a methodology and tooling for simulation-based testing
- and finally, how such an approach can be advanced, especially regarding the reuse of models (e.g. environment models) and ideally automated tests throughout the whole development process.

For this purpose, the “Simulation-based Development for Electric Vehicles” (in German: „Simulationsgestützter Entwurf für Elektrofahrzeuge“ [Shortname „E-Mobil“]) project was

set up. On the one hand it aims at finding the most suitable ECU abstraction. On the other hand, we want to establish a proper tooling that best fits the needs of the early development stages. A recuperation scenario, which is a common test scenario in the electric vehicle domain, is used for evaluation. This paper presents our current research and the first results (the project runs until October 2013).

The following section will start with an overview of how ECUs can be abstracted on the architecture level.

2 State of the Art

2.1 ECU Architecture Abstraction

In section 1 it was already suggested that V-ECUs can be used to test both the application software and some of the non-application-related code without using the actual ECU hardware. Therefore, the V-ECU must adequately abstract an ECU. The first question is to which degree the ECU can be abstracted. The second question is if a certain level of abstraction can be considered 'adequate' for early stages of development. There is a trade off between a minimized level of abstraction and the ability to simulate even if an in-depth knowledge of the final hardware as well as the final production code is not available. E.g., if the basic software is to be simulated on the lowest possible level of abstraction, the production basic software must be available as binary object code. The most accurate simulation for this would be a cycle-accurate simulation of the microarchitecture. This makes little sense if the target hardware is not known and if not the whole ECU production code (the basic software alone would not be enough) is available. If, on the other hand, the application software C code is to be tested in combination with just one aspect of the basic software, like a diagnostic event manager (DEM) on configuration level, a much higher level of abstraction is needed.

In a first step, we analyzed different levels of ECU abstraction. This section presents our findings in the form of a classification. Two things should be noted. First, we concentrate on abstracting AUTOSAR ECUs, which is required in our project. Second, we remain on the architecture level. Lower levels like the digital logic, circuit and physical implementation are not considered, because they have no influence on the development of the software.
Each ECU abstraction is modeled on a certain platform. In this context, a platform is characterized by an execution environment that offers an interface which hides implementation-specific details of the platform to the model (adapted from [17]). For example, MATLAB®/Simulink® as a platform allows modeling functional behavior, using a graphical block diagram. In model-based development, the analysis of the functional behavior is done using a specification model, ideally without the impact of any implementation specific aspects. This is classified as the fifth level of abstraction, which is depicted in the last column of table 1. The software component (SWC) is simulated, but the model uses means of computation that are different from the later production code (e.g. it uses floating-point math as opposed to integer math).
An example for a level five ECU abstraction is a model of a controller in MATLAB®/Simulink®, which specifies the desired functional behavior of an ECU.

abstraction level	1a	1b	2	3	4a	4b	5
SWC	p	p	p	p	p	p	s
RTE	p	p	p	(s)	s	s	
BSW	p	p	p	(s)	s		
OS	p	p	s	(s)			
MCAL	p	p	s				
ISA	s	s					
μ-arch.	s						

Table 1: The table shows if aspects are not considered (empty), simulated (s) or if the production (p) implementation is used in different levels of the architecture abstraction of AUTOSAR ECUs.

The simplest approach to analyze the functional correctness of the implementation of the hardware-independent application software under test (SUT) is to execute it on the host CPU of a standard PC running a common desktop operating system. Thus, platform-specific properties, such as computation time, can not be considered.
For an open-loop unit test, it is enough to trigger the application SWC and to access signals or at least the elements of its interface. However, the typical purpose of an embedded system is its interaction with an environment. Thus, to analyze the dynamic behavior of the application software, it is necessary to connect it to an environment model. Solutions exist that help to connect the application software to environment models. They also provide an interface to access signals and to trigger the execution of the application software as well as the environment model.

The standardization of application software components by AUTOSAR allows the formal description of the software components' interfaces. Based on this software component description and additional information[1], the AUTOSAR methodology [2] enables the automatic generation of glue code, which encloses the application software components. In AUTOSAR the glue code is called the Runtime Environment (RTE). Besides generating code,

[1]The RTE is generated based on the ECU configuration description, which references the used software component descriptions.

tools like dSPACE TargetLink use the software component description to create an adapter to the execution platform automatically (level 4b in table 1). This mechanism is used to embed the software-in-the-loop-simulation of dSPACE TargetLink in MATLAB®/Simulink®. In this SIL simulation the environment model is still in MATLAB®/Simulink®, while TargetLink generates a wrapper to embed the implementation in MATLAB®/Simulink®. If software components depend on AUTOSAR services, e.g. on the AUTOSAR NVRAM Manager to access non-volatile data, it is necessary to provide a simulation of the needed standardized AUTOSAR services (level 4a in table 1). However, the triggering of the runnables is different from the one of the target system, because there is no simulation of the operating system on this level.
To analyze the interaction of multiple runnables, a more precise triggering is required. This is achieved by simulating the operating system of the ECU on top of a PC running a common desktop operating system (level 3 in table 1). Operating systems of ECUs are standardized by the OSEK-OS [19] specification. The operating system module of AUTOSAR [3] is based on OSEK-OS. Furthermore, there are solutions that extend the simulation of the operating system with I/O, especially automotive networks. This is necessary to connect the ECU model to an environment model.

The following approach aims at emulating the hardware-specific layers (level 2 in table 1), which is the operating system and the hardware-dependent microcontroller abstraction layer (MCAL) of the AUTOSAR basic software. Here, the production code of all hardware-independent basic software modules is used for simulation. This means that the simulated AUTOSAR basic software very much resembles that of the real ECU.
Note that before the simulation can be run, it is necessary to configure the whole AUTOSAR basic software using valid parameters. In exchange, this approach enables analyzing basic software modules and their configuration in combination with the application software. This approach can be realized both on consumer desktop PCs and on a real-time prototyping platform to connect the I/O of the ECU model to the real environment.

Other approaches simulate parts of the basic software and hardware, particularly automotive networks, in system-level design languages, like SystemC. In [14] the mapping of the AUTOSAR architecture on SystemC is described. This work is focused on modeling the timing of a communication connection. [5] concentrates on modeling the timing of a single ECU. The execution times were measured using a real evaluation board and then used in the simulation. In [4] the dynamic execution times are estimated by an extension of QEMU[2]. In all this approaches, the operating system is emulated by a model of a real-time operating system that has been designed in SystemC as well.

[7] presents an approach to simulate only the hardware, especially the microprocessor, of an ECU (level 1 in table 1). This can be achieved in two different ways. One is the Instruction Set Simulation (ISS) where the instruction set architecture (ISA) of the target microprocessor is simulated (level 1b in table 1). The ISA includes native data types, instructions, registers, addressing modes, interrupts, exception handling and external I/O. The simulator is able to execute the production binary object code of the ECU software, including the operating system.
A Cycle Accurate Simulation (CAS) even simulates the microarchitecture of the target processing unit. This, as already mentioned in the introductory example, is the lowest level of abstraction (level 1a in table 1) that we consider in this classification. The CAS respects the exact timing behavior of the ECU, because it includes the simulation of caches, pipelines, branch prediction and other details of the microprocessor.
All of the presented levels of ECU abstraction can be used for simulation-based testing. To act as a link between early simulation of pure application software simulation and HIL simulation, it would be advantageous, if the tests that are developed for early behavioral simulations, could be reused across the different levels of abstraction. This ideally includes 'level 0' (the ECU prototype) that is used later in the development process. The next section gives an overview of the current research situation on this matter before section 3.1 uses the presented classification to find the most suitable ECU abstraction for our purpose. This essentially boils down to the question, which of these levels are relevant for the early phases of electric vehicle development.

2.2 Reuse of Tests Across Domains

To increase the benefit of using abstract models of ECUs for early simulation-based testing, it is one major goal of our project to find ways of modeling tests that can be used through different levels of ECU abstraction (from the behavior specification to a real ECU) seamlessly. This chapter presents some research that will help us to develop our own approach later in the project. As a first step, we had a look at standardized ways of software testing. The international standard ISTQB, for example, formulates seven principles. Principle three suggests that testing should be started as early as possible [6]. This supports our claim of frontloading tests using ECU abstractions.
The model with the highest level of abstraction (level 5 in table 1) is the first testable model in the development process. Here, the set of possible faults is the smallest. Generally, it holds that the lower the level of abstraction, the more information is required. This information may not be defined in an early stage of the development process. For example, an ECU of abstraction level 2 requires an in-depth

[2] http://qemu.org/

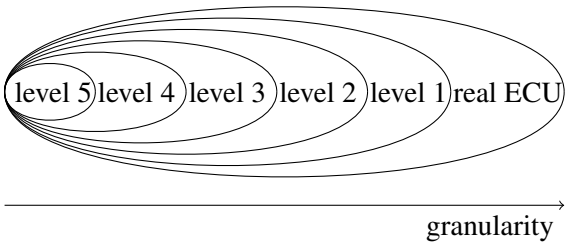

Figure 1: The granularity of abstraction different levels of ECUs.

knowledge of the configuration of the basic software to enable the simulation of a concrete implementation. In exchange, a platform that models a low ECU abstraction level enables the detection of many more faults. These include the faults that are introduced through adding more detail, but also those of higher levels of abstraction. Figure 1 visualizes this relation. Different platforms are suitable to analyze and test different aspects of the target software.

The transition to a model with a lower level of ECU abstraction is an enrichment of details. To ensure the correctness of the additional details, back-to-back testing is used. Back-to-back testing means testing the model of higher abstraction level and then testing the model of lower abstraction level using the same test cases and scenarios, including the same environment model, and comparing the results. ISO 26262 [13] recommends back-to-back tests for all automotive safety integrity levels (ASILs). dSPACE TargetLink already provides back-to-back tests of the model (level 5) and the application implementation (level 4b). It is necessary to ensure that the implementation (level < 5) adequately approximates the specification model (level 5). Remember that it may now be triggered by an operating system (level ≤ 3) or relies on input data provided by other components such as communication (level $\leq$ 4a). Finally, it is necessary to test on the real hardware too, because there is currently no ECU abstracting model that represents all aspects of a real ECU.

An ideal test system is able to apply the same test to the behavioral model, to any other level of ECU abstraction and to the real ECU. If feedback from the output of the tested artifact through an environment model is needed, this means that the same environment model must be used in all test scenarios. For back-to-back testing it is a further requirement that the tests can be reused on different platforms. This increases the quality and flexibility of the test case and reduces the costs of recreating them.
Each level of abstraction (table 1) uses its own representation of values. The level of the behavioral model (level 5) uses physical values, which are often normalized, but independent of their technical representation. The software implementation levels use a concrete technical representation of the values, e.g. a scaled integer in a logical communication data unit, like a binary CAN frame.

Finally, the real ECU is interfaced with real analog or digital ports that are vulnerable to physical effects such as electrical disturbances. The diversity of platforms requires that the reusable test has to be modeled in a way which is platform-independent. This platform-independent specification of a test is called an abstract model of an test. Early approaches to model tests on a high abstraction level can be found in [20] and [12]. In [20] test descriptions are generated from a formal test specification, with the focus on data abstraction.

To execute these tests, they have to be mapped on a concrete platform. One way of describing such a mapping is given by [12]. This standard defines the "Tree and Tabular Combined Notation", which models data formats, behavior and interfaces on a high abstraction level and handles the mapping of interfaces to real systems.

One technique to model a test on a high abstraction level in an platform neutral way is presented by [9]. They use the unified modeling language (UML) to describe tests, which are used as a basis for generating a concrete implementation (e.g.JUnit tests). In a model driven architecture (MDA), which is described in the MDA Guide ([17]) a platform-independent model (PIM) is described. This model focuses on the application behavior, which is platform-independent. The platform-specific model (PSM) maps the PIM on a concrete platform. It is furthermore the input for the generation of the platform code. [25] combines the ideas of the abstract test modeling using UML ([9]) with the philosophy of model-driven architecture. Here they refer to an older approach presented by [23]. Therefor, they introduce two different test models: the platform-independent test design model (PIT) and a platform-specific test design model (PST). Furthermore, they describe the associated transformations. The basic idea is to transfer a methodology, which is established in software development, to the development of tests.

Another approach is to formulate a test for different levels of abstraction using a designated language. [10] presents such a test language called TestML. The goal was to exchange testing systems between the behavioral model (level 5), the implementation (level 4) and the real ECU. They proved that a test case formulated in that language could be run on a MATLAB®/Simulink®model. There was no proof in this work that other platforms can be supported as well. However, we think that this approach is promising. This is because the language is formulated on an abstract test system. They define this abstract test system to consist of "a combination of test components that [they] consider minimally necessary regarding the exchange of test descriptions." The system is separated into three components: the

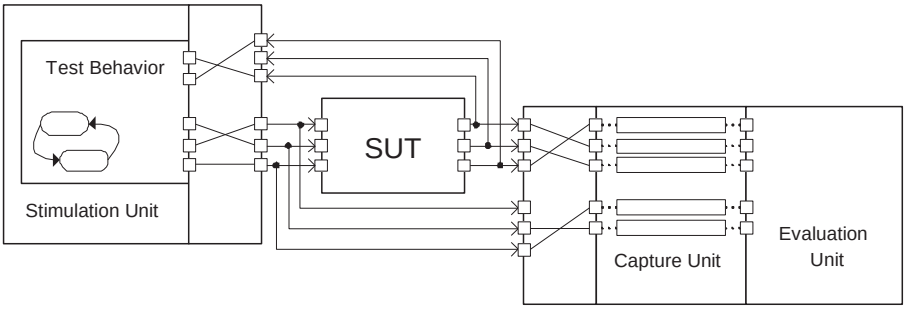

Figure 2: Abstract test system as used in [10].

test itself, the system under test (SUT) and the evaluation unit. Figure 2 shows an illustration of their abstract test system. The stimulation unit generates stimuli, but it does not yet contain any link to an environment model. The test behavior is described by hybrid automata that have been described by [16]. They are able to model timed applications and systems with discrete and continuous behavior. The capture unit is responsible for recording the system reaction and the stimuli. The verdict is provided by the evaluation unit.

Anyway, the key is the description of the interfaces between these components. They abstract the necessary link between the SUT and the other components. This means that every platform only needs a mapping for the interfaces and not necessarily the whole system. Thus, although the work did not mention any other supported platform besides MATLAB®/Simulink®, it seems that not much effort is required to extend the support to other platforms. However, in our project it is explicitly required to reuse the environment model on different platforms. The TestML approach was not specifically designed for in-the-loop tests. Here, it would be necessary to enhance the stimulation unit so it can hold an environment model. That is why we intend to combine the platform-independent approach of [10] with the model-based idea of [25] mentioned earlier. With such a system it would be possible to reuse tests on all levels of ECU abstraction as well as on the real ECU. However, the origin of such a test has not been addressed yet. The test is usually derived from a textual representation of the corresponding requirement. The MDA Guide refers to this as computation independent model (CIM) – "a view of a system from the computation independent viewpoint."[17]. These requirements are usually managed in tools like IBM Doors. [11] shows, that it is possible to create a formal model (SysML model) just from textual requirements. Therefore the requirements must be given in a language that is natural but also restricted in terms of vocabulary and grammar. This is commonly known as controlled natural language. In this special case a language called 'requirement patterns' is used. A requirement formulated in that language can be analyzed by extracting a subsystem and function hierarchy including inputs and outputs and using this information to create a model in SysML. This representation of the requirement creates a formal model in the CIM. This could probably be used as reference model to describe mappings of interfaces across all levels of ECU abstraction and the real ECU. However, further research is required here. Fortunately, this section has shown that the reusability of tests does not depend on the ECU model. Thus, there is no need for a special requirement regarding the choice of a suitable level ECU abstraction in the next section.

3 Our Approach

3.1 A Suitable ECU Model

Before we can answer the question of which of the ECU abstraction levels presented in section 2.1 is most suitable for our purposes, we have to reflect upon the requirements we have to satisfy. Our primary goal in the 'E-Mobil' project is to enable the simulation of application code behavior under real ECU conditions in early development phases. This first and foremost requires a separation of the application-related and the non-application related software running on the ECU. Imagine the developer works in the application layer. Ideally, it should not be necessary to care about lower layers like the operating system or network communication protocols. In early stages of development, the developer has no or only a little knowledge of those. In spite of that, the application software should be able to use services of lower layers as if they were already implemented. This way, a high level of integration can be achieved without the need for the final basic software. Furthermore, interaction between several application software components should be possible.
Another typical use case is that some configurations of the system architecture (e.g. a BSW module or a communication behavior) are already known (but not fully implemented). It should be possible that the application software can be simulated in an environment that emulates these aspects.
In both cases, there have to be well-defined interfaces between the application software and the

non-application relevant software.
The AUTOSAR standard [1] defines those interfaces. They are called basic software (BSW). Furthermore, AUTOSAR defines a layered and modular software architecture for ECUs. Regarding section 2, this enables the simulation of the ECU on different levels of abstraction.
For closed-loop simulations, the ECU model must provide means to connect to an environment model.
To use the simulation for testing, measurement and calibration support is obligatory. This applies for both, the environment model and the ECU model. Here, the variable measurement of the ECU model should be as close to that of the later ECU prototype as possible.
In summary the requirements can be formulated as follows:

- The ECU model should be able to include and simulate AUTOSAR application software components (SWCs), so the model of the ECU must have an abstraction level lower or equal than 4b.
- It should be possible to simulate multiple SWCs on a virtual functional bus (VFB) level. This requires an RTE simulation which is able to connect multiple SWCs.
- To simulate the effects of communication between different application software components, an RTE implementation must exist that abstracts intra- as well as inter-ECU communication.
- For the inter-ECU communication a bus simulation as well as the simulation of the responsible BSW modules (COM Stack) must exist. BSW modules are considered in abstraction levels lower or equal than 4a.
- To simulate realistic task behavior and task triggering, the V-ECU must include an AUTOSAR OS implementation, which requires an abstraction level not higher than three.
- For open- and closed-loop simulation, the model must provide interfaces to an environment model. These interfaces should be either bus interfaces or signal interfaces that mimic the later I/O of the real ECU.
- Services for experimentation, debug and test support should be included. The most important service is the XCP service that complies with the ASAM MCD-1 MC standard.
- It should be possible to analyze and simulate the application software with aspects of basic software that only require some configuration (and no final implementation).

With these requirements in mind, it was obvious that we need an ECU model that spreads across several levels of ECU abstraction (see section 2.1). Yet, we found that on the lower end, a level 3 solution is sufficient here. Level 2 and level 1 seem to be too close to the real ECU prototype, as they assume a comprehensive knowledge of the final hardware and that a full configuration of the basic software is available.
Anyway, it appears that the virtual ECU (V-ECU) presented by [21] satisfies our requirements. It allows a simulation of the application software without the need to care about basic software. However, if the developer is interested in realistic task behavior and task triggering, he/she is able to configure the operating system and the RTE. Furthermore, the V-ECU provides an implementation of the AUTOSAR communication layer (COM) and services of an AUTOSAR basic software like a diagnostic event manager (DEM) and an ECU state manager (ECUM), too.
In our opinion the V-ECU can act as a link between the pure application software and the ECU prototype, between simulating the application code individually and a HIL simulation. Together with the functional model, this results in four concrete model characteristics that an ECU runs through. These are depicted in Figure 3. One can see which aspects of the ECU can be tested on which model characteristic and what type of simulation is used.

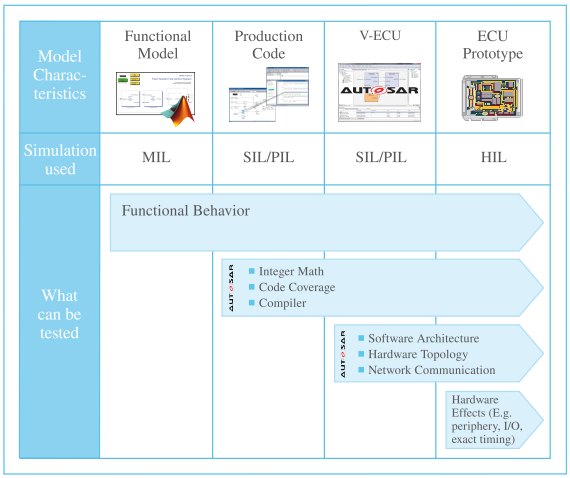

Figure 3: The virtual ECU in the context of simulation-based testing.

The next section presents a suitable tool chain that integrates these characteristics except the ECU prototype and their corresponding simulation types.

3.2 Requirements for a Tooling

The main benefit of using V-ECUs in the context of developing electric vehicles lies in early simulation-based testing. Compared to a HIL simulator that requires a close-to-production ECU prototype, the virtual ECU allows for simulation even if many aspects of the production ECU are still unknown. This, as already mentioned in section 1, is especially beneficial for the development of electric vehicles.

However, both modeling and simulating the different levels of ECU abstraction (see section 2) when developing an ECU from scratch, as well as the architecture and first non-functional aspects demand a good tooling. Our observation is that although the application field of many commercially available tools is expanding continuously, there will not be a single tool handling all these aspects adequately in the foreseeable future. Instead, there is always quite a number of specialized tools that form a dedicated tool chain. These tools' capabilities are employed by the domain they are used in. We see domains as separate groups of developers that, due to their particular task in the development process, are specialized in different areas and therefore have different technical knowledge and orientations. For the early stages of development we consider mainly three domains involved: The function developer domain, the software architect domain and the tester domain. A function developer usually has a very different background than a software architect. The former focuses on mathematical modeling of control algorithms whereas the latter is specialized in software development. Hence, our goal is to establish a methodology across the named domains and an adequate simulation-based tool chain that is particularly well suited to design, implement and test software for electric vehicles in early development stages. As a consequence, we impose certain requirements:

- Every domain should be able to use simulation. This ranges from open-loop simulation using simple stimulation to in-the-loop simulation using a designated environment model as a means for validation, testing and/or optimization. In this context it is important to note that an environment model can be (re-)used across multiple domains (and therefore across different tools).

- The tool chain should provide means for experimenting and testing these early artifacts in an environment that is commonly used in HIL scenarios. This allows frontloading of tests as described in section 1.

- The tool chain must handle AUTOSAR-compliant software. AUTOSAR facilitates the exchange of the software components between tools and domains and furthermore separates the functional aspects (application software) from the non-application relevant aspects (RTE and basic software).

- The tool chain should feature tool-assisted transitions beyond tool boundaries and between the three domains mentioned above. This includes the reuse of artifacts across multiple domains (e.g. the plant model) as well as an appropriate model conversion/import/export. For example, between the function developer and the system architect there is a point of contact where the controller functionality gets integrated into the AUTOSAR-compliant architecture. Here, we have domain-specific tools on both sides. This bears a risk of tool gaps which must be bypassed.

- The tool chain should allow for model-based or at least model-driven design and development. This, despite the fact that there is still plenty of hand-written code even in recent ECUs, is imposed by the growing complexity of automotive software paired with high safety standards. Furthermore, it enables changes on a prototyping basis.

- The tool chain should assist handling the general problem of how true universal tests that originate from requirement management tools (e.g. DOORS) can be used across domains. This means that one test can be performed on a MATLAB®/Simulink®model, on a V-ECU and ideally even on a real ECU, for example.

The next section shows how we plan to satisfy these requirements. Therefore, we use state-of-the-art tools within a tool chain that extends across multiple domains.

3.3 The V-ECU Simulation Tool Chain

In order to meet the requirements presented in section 3.2 we first established a basic tool chain that will be continuously enhanced throughout the project. For this purpose, it is going to be used by us and our project partners to implement a scenario, that is characteristic for electric vehicles. We chose a recuperation scenario as recuperation is a key feature of electric vehicles. Our project partners will represent the members of the different domains. This allows us to evaluate the tool chain, especially regarding unexpected use patterns and tool gaps. In the following it is explained how this tool chain can be used and how we plan to further enhance it.

In the beginning the function developer designs the first behavioral models as a specification using MATLAB®/Simulink®, which can be seen in the upper third of figure 4. He or she uses model in-the-loop (MIL) simulation for validation (also compare with the first column of figure 3 in section 1). Therefore an environment model has to be created. Our environment models are based on the dSPACE Automotive Simulation Models (ASMs). These already include a multi-body system of a car, customizable maneuvers, roads and ready-to-use models of electric components. Due to the fact that these models are actually intended for real-time simulations in a HIL context, they offer good performance. Another big advantage is that they can be easily parametrized using the GUI-based parametrization environment ModelDesk. Furthermore, it is possible to add own models to the ASMs and parametrize them in ModelDesk as well.

After the behavior and the environment have been modeled, the former must be transformed

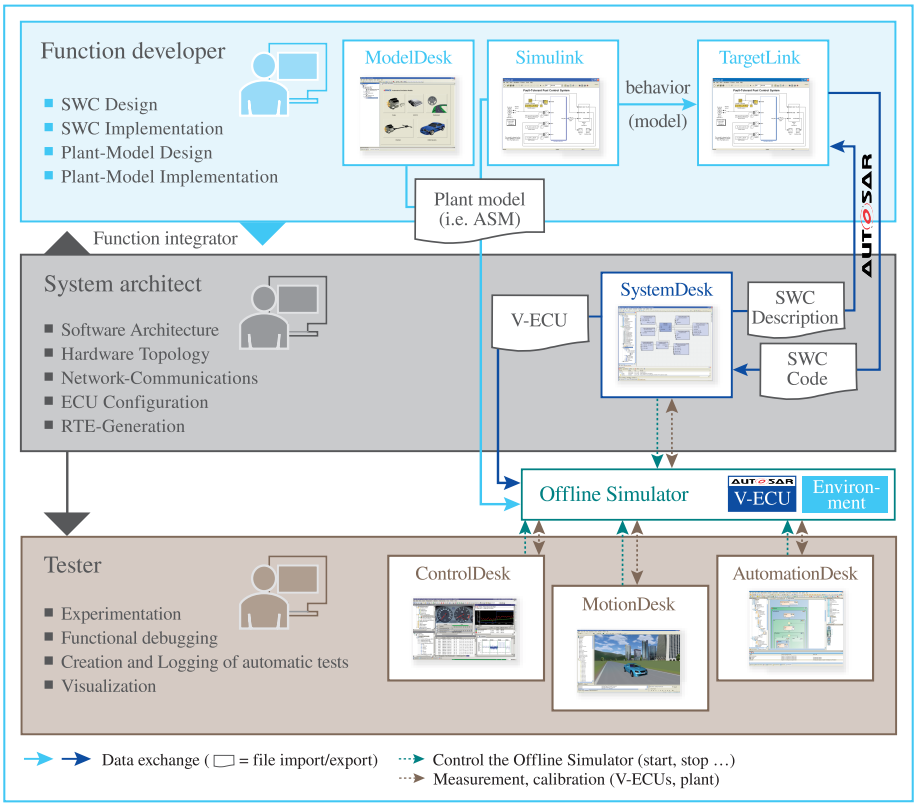

Figure 4: The three main domains and the tool chain.

into actual code. The transformation into an implementation model can be done automatically by tools like TargetLink. Therefore, it converts the Simulink blocks into special TargetLink blocks. It should be noted though that not all Simulink blocks can be converted, as the code generation and production-capable ECU code itself by nature imposes certain limits. Here, the function developer has to keep in mind that through this transformation, continuous models will be discretized with regard to constants, interim results and signal data that are now represented as integers instead of floating-point numbers. Not to mention the fixed-step nature of the solving mechanism itself.

After the implementation model is generated, the software developer has to configure bit widths and scalings. However, this normally depends on the target hardware (e.g. an 8-bit ECU or a 16-bit ECU) which might not be known at that time. Hence, the software developer could just use any configuration that sufficiently approximates the functional model.
With the generated production code, the function/software developer can use software in-the-loop (SIL) or, if a PIL device is available, processor-in-the-loop simulation, also seen in the second column of figure 3 in section 1. The new behavior is compared with the results from the earlier MIL simulations. Note, that the plant model is not affected by the code generation. In fact, the plant model being available to all domains is one major requirement for universal tests. A more detailed analysis is given in section 2.2.

Alongside the functional models, the system architecture is being designed or adapted to facilitate the integration of the application code. This is depicted in the middle of figure 4. Note that no chronological order is implied here: e.g. the function developer and the system architect both can start at the same time.
The system architecture includes the software component frames (ports, interfaces), the configuration of runnables and any (re-)configuration of non-application-relevant aspects (RTE, basic software, the network communication, ...). Note, that we distinguish two use cases here: On the one hand the creation of a system architecture from scratch which is used as a starting point for the simulation of functional code together with non-functional code, on the other hand the integration of new (or modified) function code into an existing V-ECU topology. Both use cases are common for the early development stages of electric vehicles and both appear to be complex tasks. In our project we use SystemDesk to assist the system architect in that matter. SystemDesk was created with system architectures in mind and is especially targeting AUTOSAR-compliant architectures. It supplies the user with the means to compose AUTOSAR software components with ports and interfaces and the respective data elements in a model-driven fashion.

The software components can be mapped on the modeled ECUs. Then the OS as well as other basic software (BSW) modules like the COM module can be configured. Furthermore, the network communication (IPDUs and signal mapping) and an automotive bus can be modeled. The code for the AUTOSAR Runtime Environment that acts as a communication center for inter- and intra-ECU information exchange on a logical level can be generated automatically.

Out of this data SystemDesk generates the V-ECU executable, which can be simulated with the dSPACE Offline Simulator. The Offline Simulator can simulate several V-ECUs in combination with an environment model. The latter originates from MATLAB®/Simulink®, where it has automatically been converted into a simulation executable. The simulation is performed on a standard PC. As already mentioned in section 1, the V-ECUs are not simulated in real-time but as fast as possible on the underlying PC hardware. Note that the simulation speed often only depends on the environment model, which usually uses much more computation time compared to the ECU code.

The Offline Simulator is depicted between the system architect domain and the tester domain on figure 4. A detailed description of how tools like SystemDesk enable the user to integrate the existing software into a V-ECUand how they are simulated with the Offline Simulator can be found in [18].

Certain questions that arise are: How can the function developer create the ECU functions according to the structure and interfaces that are developed in the system architect's domain? How, on the other hand, can the system architect be sure that the functional code will fit into the prepared V-ECU? How does the environment model get integrated into a simulation with a V-ECU topology? A transition between both domains as mentioned in section 3.2 must be realized. The answer is quite easy for the environment model. As mentioned earlier, it can be automatically converted into a simulation executable, which can be simulated with the Offline Simulator. Also, it is possible to connect the environment model with a designated port on a V-ECU. However, the transition of the system architecture is not that easy. A simple exchange of AUTOSAR files is not enough. The function developer might lack a certain expertise when it comes to AUTOSAR. What he or she needs is a representation of the SWC frame as a TargetLink frame that somehow includes the necessary structures mapping to the corresponding sender/receiver or client/server data access points. TargetLink offers a data dictionary that imports and stores the AUTOSAR information. It provides mechanisms to create a frame of model blocks that already has the appropriate ports and signals set up. The function developer only needs to put in the functionality for which the code is to be generated. The code generation of TargetLink implements the RTE calls according to the system description. This way it can easily be integrated by the system architect.

Nevertheless, there are some challenges that have yet to be addressed. When a system is created from scratch, many aspects of the architecture such as the actual interfaces between software components are still unclear due to the fact that some requirements may change later. The system architect then needs the function developer to adapt the models to the new interface to obtain an updated implementation. This presupposes that the architectural changes are propagated into the MATLAB®/Simulink®model as well.

The last part of the tool chain are the experimentation and testing tools, which can be seen on the lower third of figure 4. The V-ECU, like a real ECU, supports measurement and calibration according to the ASAM MCD-1 MC (XCP) standard. This means that ASAP2 files (according to ASAM MCD-2 MC standard) are used to describe the mapping of the variables. This is important, as common HIL test and experimentation tools need these variable descriptions. For measurement and calibration, ControlDesk Next Generation is used. Automatic testing is done using AutomationDesk. MotionDesk is used for plausibility tests as it renders the simulated ASM model as a 3-D animation. Furthermore, ControlDesk Next Generation and AutomationDesk both support the Offline Simulator. The Offline Simulator in turn can simulate V-ECUsand models originating from MATLAB®/Simulink®. Hence, experimentation and testing across multiple domains is possible.

4 First Results and Future Plans

The project is still quite at the beginning. In the first stage of the project we mainly concentrated on the function developer domain. Together with our project partners, the first behavior models for the recuperation scenario were created. At the same time, after having established the methodology and requirements presented in section 3.2, some tool enhancements have been created.
Until now, the main focus laid on the transition between the function developer and the system architect domain. There, we developed a concept and a prototype implementation for an update functionality of the model generator that creates a TargetLink subsystem (MATLAB®/Simulink®model blocks) from an AUTOSAR description. Here we link the AUTOSAR data with the related TargetLink blocks so that the implementation model is 'aware' of its underlying AUTOSAR architecture. One challenge here is, for example, to establish the link so that it is robust against changes in both the TargetLink subsystem and the AUTOSAR description. Furthermore, there should be no need for additional files besides the AUTOSAR description and the TargetLink model that have to be transported across domains. We expect the update functionality to be frequently used (and therefore evaluated) during the next months when the behavioral control

models (energy management, engine control, brake control) are transformed into V-ECUs.
Besides that, we have already started to develop means to ease up the work of a system architect and increase the value for offline simulation by introducing first concepts of new ways to analyze an AUTOSAR-based V-ECU topology. Simultaneously, further research will be done on easing up the configuration of V-ECU topologies paired with a consequent use of model-driven design principles. Our goal here is to reduce the time needed to create configurations ready for simulation that enables the development and modification of V-ECU topologies on a prototyping basis.

Furthermore, the field of reusability of tests across domains will be addressed in detail. The final goal here is to allow for the definition of tests on a semantic basis that can be executed on various platforms. Even though the requirements that form these tests are expressed as natural text, e.g. in DOORS, they are in the view of the computation-independent model. We hope to use these findings to create links between test cases, environments models, implementation artifacts and requirements going beyond traceability. Finally, we plan to implement the means to create reusable tests and enable our project partners to evaluate them.
At the end of the project, together with our project partners, we are going to have a running demonstrator for the recuperation scenario. Here, the control functionality will be realized as a V-ECU and will be simulated together with environment models from MATLAB®/Simulink®in an offline simulation environment. As stipulated in section 3.2, HIL test tools will be used for evaluation. Furthermore, we aim at testing some of the control functionality on a prototype device against a real test bench.

5 Conclusions

In this paper we presented our research regarding the early simulation and testing of virtual ECUs for the development of electric vehicles. We started with a brief presentation of the idea behind virtualizing ECUs and presented a classification of recent approaches.
We then formulated the requirements and assumptions on the V-ECU itself as well as on a possible tool chain that allows frontloading of tests which usually can only be performed in later HIL setups. We identified the domains involved and their points of contact.
These considerations lead to the methodology and tool chain described in section 3.3 that we use as basis for our project. Here we examined where the tool chain can be further enhanced to avoid tool gaps and to accelerate the development process in early stages of development.
Particular attention was paid to reuse of tests across domains. It has been shown that there are already promising approaches that we will follow to allow for the creation of tests for various platforms.
Last, but not least, we presented the first results and provided a glimpse of our plans for the upcoming project phase.

Acknowledgments

The presented work is funded by the European Union and the state of Nordrhein-Westfalen (grant No. 64.65.69-EM-1008A). The project is conducted together with the University of Paderborn and the DMecS GmbH.

References

[1] AUTOSAR (automotive open system architecture). http://www.autosar.org/.

[2] Autosar. AUTOSAR Methodology, 2008.

[3] Autosar. Specification of Operating System, October 2010.

[4] Markus Becker, Henning Zabel, and Wolfgang Mueller. A Mixed Level Simulation Environment for Stepwise RTOS Software Refinement. Technical report, University of Paderborn/C-LAB, 2010.

[5] Markus Becker, Henning Zabel, Wolfgang Müller, and Ulrich Kiffmeier. Integration abstrakter RTOS-Simulation in den Entwurf eingebetteter automobiler E/E-Systeme. In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Berlin, Germany, March 2009.

[6] International Software Testing Qualificytions Board. Certified Tester - Foundation Level Syllabus, 2011.

[7] Mikaël Briday, Jean-Luc Béchennec, and Yvon Trinquet. Modelisation of a Distributed Hardware System for Accurate Simulation of Real Time Applications. In *Proceedings of 5th IFAC International Conference on Fieldbus Systems and their Applications (FeT'03)*, july 2003.

[8] Rainer Dömer, Andreas Gerstlauer, and Wolfgang Müller. Introduction to hardware-dependent software design hardware-dependent software for multi- and many-core embedded systems. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ASP-DAC '09, pages 290–292, Piscataway, NJ, USA, 2009. IEEE Press.

[9] Hans-Gerhard Gross. Testing and the UML. A perfect fit. Technical report, Frauenhofer Institut für experimentelles Software Engineering, October 2003.

[10] Jürgen Grossmann, Ines Fey, Mirko Conrad, Alexander Krupp, Wolfgang Müuller, and Christian Wewetzer. TestML – A Test Exchange Language for Model-based Testing of Embedded Software. In *Proceedings of Automotive Software Workshop '06, Oct. 2007*, October 2007.

[11] Jörg Holtmann, Jan Meyer, and Matthias Meyer. A Seamless Model-Based Development Process for Automotive Systems. In *Software Engineering 2011 - Workshopband (inkl. Doktorandensymposium)*, 2011.

[12] ISO. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN), 1992.

[13] ISO. ISO 26262-6: Road vehicles – Functional safety – Part 6: Product development at the software level, 2011.

[14] Matthias Krause, Oliver Bringmann, and Wolfgang Rosenstiel. Verification of AUTOSAR Software by SystemC-Based Virtual Prototyping. In Rainer Dömer Wolfgang Ecker, Wolfgang Müller, editor, *Hardware-dependent Software - Principles and Practice*. Springer, 2009.

[15] Karsten Krügel and Klaus Lamberg. Virtuelle Steuergeräte als Grundlage einer frühzeitigen Absicherungsstrategie. Contribution to the 2nd Workshop "Automotive Software Engineering: Virtuelle Absicherung", February 2011.

[16] Eckard Lehmann. *Time partition testing: systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen*. PhD thesis, TU-Berlin, Berlin, 2004.

[17] Joaquin Miller and Jishnu Mukerji. *MDA Guide Version 1.0.1*. OMG, 2003.

[18] Oliver Niggemann, Anne Geburzi, and Joachim Stroop. Benefits of system simulation for automotive applications. In *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, MBEERTS'07, pages 329–336, Berlin, Heidelberg, 2010. Springer-Verlag.

[19] OSEK/VDX. Operating System, February 2005.

[20] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating fuctional tests. *Commun. ACM*, 31:676–686, June 1988.

[21] Dr.-Ing. Rainer Otterbach, Dr. Oliver Niggemann, Joachim Stroop, Dr. Axel Thümmler, and Dr. Ulrich Kiffmeier. Durchgehende Systemverifikation im Entwicklungsprozess. *ATZ - Automobiltechnische Zeitschrif*, 4:298–307, 2007.

[22] Herbert Schuette and Peter Waeltermann. Hardware-in-the-Loop Testing of Vehicle Dynamics Controllers – a Technical Survey. In *2005 SAE World Congress*, 2005.

[23] Jon Siegel and Object Management Group. Developing in OMG's New Model-Driven Architecture. 2001.

[24] Peter Wältermann. Hardware-in-the-Loop : The Technology for Testing Electronic Controls in Automotive Engineering. *Translation of the 6th Paderborn Workshop in Designing Mechatronic Systems*, 2009.

[25] Justyna Zander, Zhen Ru Dai, Ina Schieferdecker, and George Din. From U2TP Models to Executable Tests with TTCN-3 - An Approach to Model Driven Testing. In *TestCom2005*, 2005.

Authors

Lars Stockmann received his degree in computational visualistics in the university of Magdeburg. After that, he worked three years as a developer of audio related applications for mobile devices.
In 2011 he joined dSPACE and since the beginning of 2012 works as technical project manager in the research project "E-mobil".

Dominik Holler received his degree in computer science from the University of Kassel. Now he works as a concept software developer at dSPACE since 2011. His particular research interest is the reuse of tests in the software development process.

Dr. Dirk Spenneberg worked at the dSPACE as project manager for the AUTOSAR architecture tool SystemDesk from 2008 till the end of 2011. Furthermore, he was one of the initiators and leading coordinator of the research project "E-Mobil".
He now works as a project manager for Bombardier Transportation.