prostep IVIP

White Paper

Virtual Electronic Control Units

prostep ivip White Paper

Smart Systems Engineering

Requirements for the Standardization of
Virtual Electronic Control Units (V-ECUs)

Table of Content

Figures

1 Introduction and background

The prostep ivip Association is a globally active, independent network comprising manufacturing industry, IT vendors and service providers, and the research community. The primary focus of its work lies in the digital transformation of the product engineering and production processes. It formulates and bundles the requirements of manufacturers and suppliers in the manufacturing industry, defines standards and interfaces, provides IT vendors with forums for improving interoperability and carries out vendor-independent benchmarks.

The prostep ivip Association's "Smart Systems Engineering" (SmartSE) project group comprises participants from almost 30 companies and research institutions. SmartSE develops application-oriented concepts for mastering the common challenges posed by Systems Engineering (SE). The association formulates recommendations for process design, drives technical standards for the collaborative development of complex mechatronic systems forward and encourages improved transparency for systems engineering objects. The project is currently focusing on the interoperability of simulation standards in the context of autonomous systems (see Figure 1).

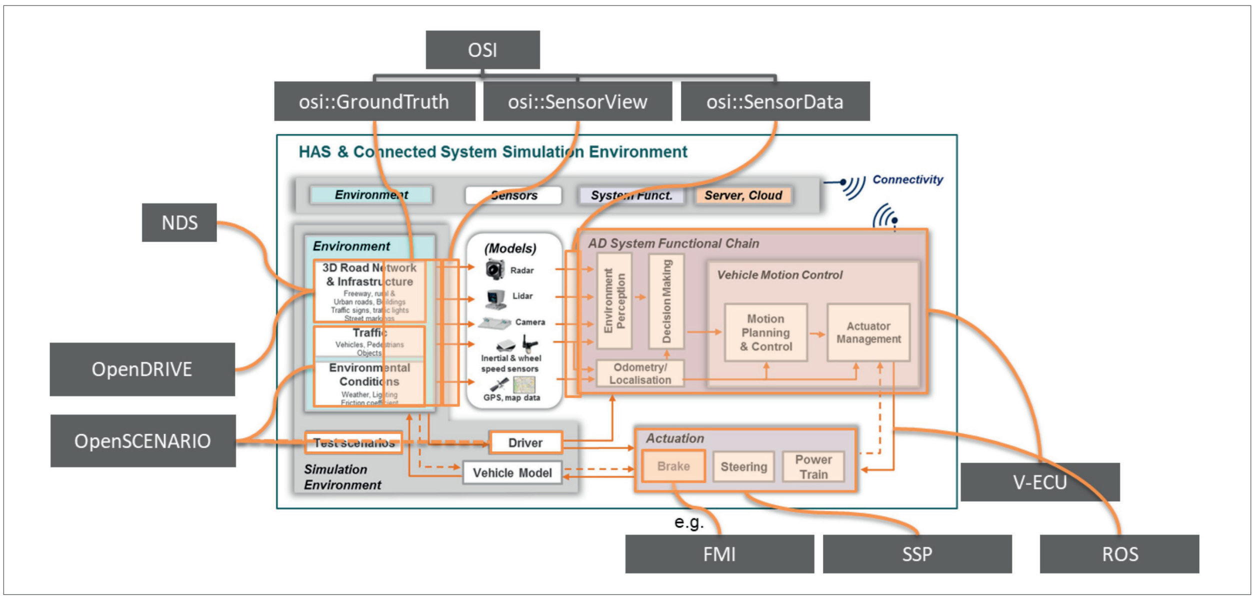


Figure 1: Simulation standards in the context of autonomous systems

Earlier in the SmartSE project, a survey of OEMs, suppliers and IT vendors was conducted with the aim of identifying and analyzing model types and simulation tasks for using these model types. One finding of the survey was that a simulation of Electronic Control Units (ECUs) in combination with models of plant and environment is an important simulation task. But instead of using the real ECUs within such simulations, it would be more reasonable to use virtual versions of ECUs, so called virtual ECUs (V-ECUs).

The project group identified a simulation task “Coupling of models of ECUs (V-ECUs) among each other and with environment/plant/vehicle models” as a topic to be prioritized regarding the potential of standardization (see Figure 2). Thus, a new work package entitled "Use Cases and Requirements for a new V-ECU Standard" was launched within the SmartSE project, with the aim of deriving appropriate use cases and requirements for a possible new V-ECU standard or extending existing standards such as FMI or AUTOSAR.

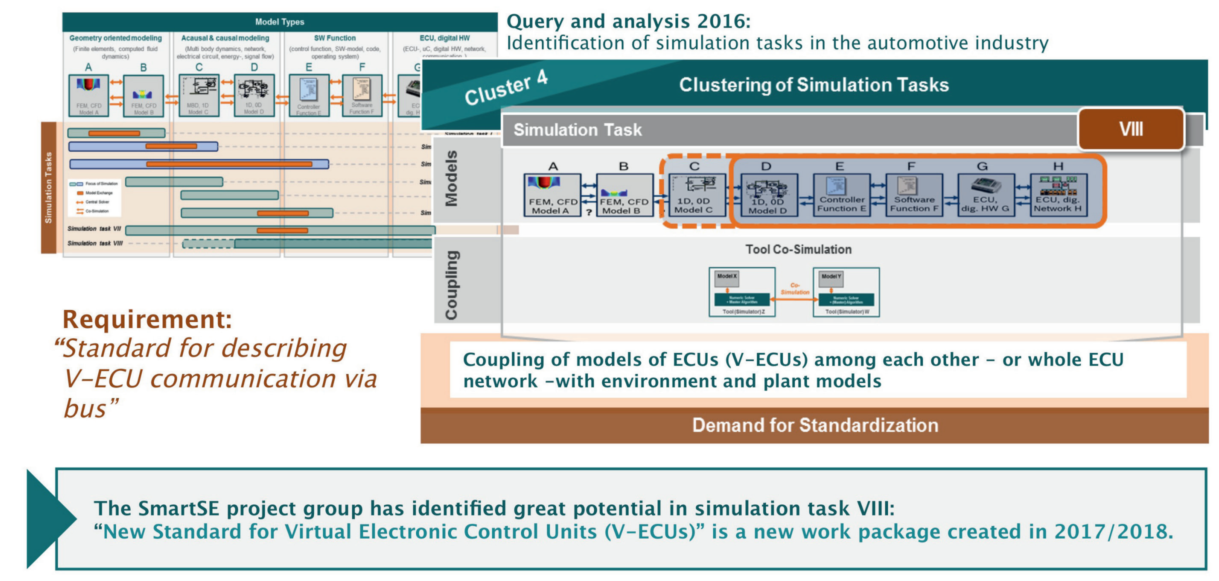

Model Types
Query and analysis 2016:
Identification of simulation tasks in the automotive industry
Cluster 4
Clustering of Simulation Tasks
Simulation Task
VIII
Models
A FEM, CFD Model A
B FEM, CFD Model B
C 1D, 0D Model C
D 1D, 0D Model D
E Controller Function E
F Software Function F
G ECU, dig. HW G
H ECU, dig. Network H
Coupling
Tool Co-Simulation
Simulation Tasks
Requirement:
“Standard for describing V-ECU communication via bus”
Coupling of models of ECUs (V-ECUs) among each other – or whole ECU network –with environment and plant models
Demand for Standardization
The SmartSE project group has identified great potential in simulation task VIII:
“New Standard for Virtual Electronic Control Units (V-ECUs)” is a new work package created in 2017/2018.

Figure 2 Motivation for the V-ECU work package

Additional surveys of the project partners addressing the use cases and the challenges posed by V-ECUs have been carried out. The key information of these surveys is shown in Figure 3.

Note:
The names of the V-ECU levels used in Figure 3 are the names that were used in the survey and not the names proposed in this paper. The new approach for the definition of the V-ECU layers can be found in chapter 4. In this new definition, only the level names have been changed; the content of the different levels has not changed.

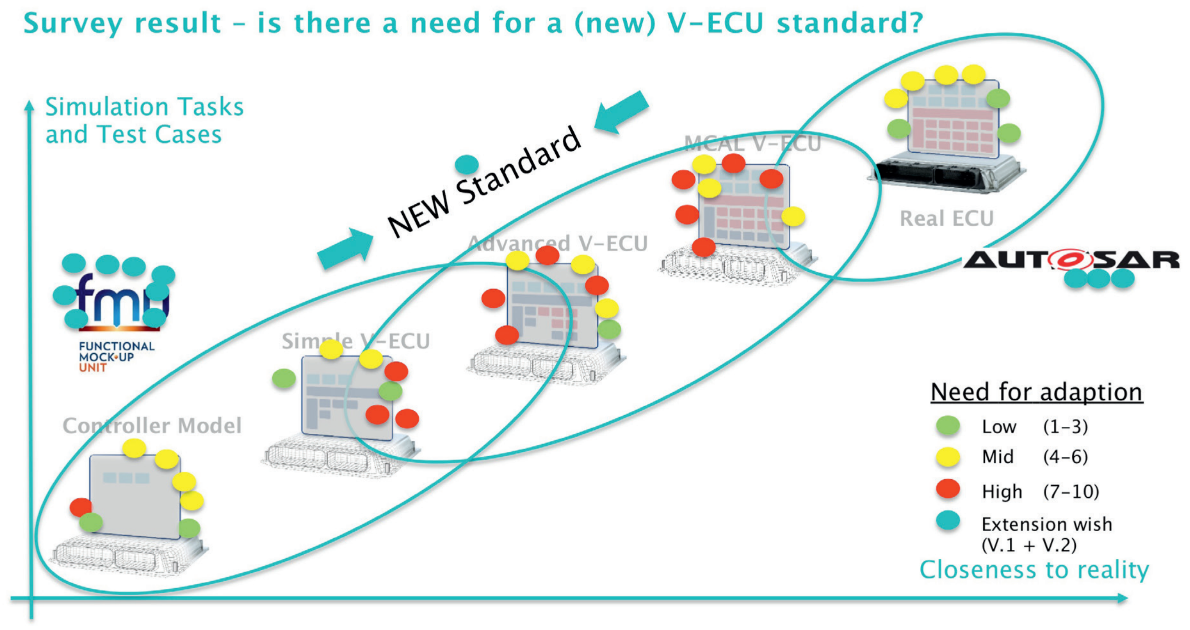


Figure 3 Need for a (new) V-ECU standard (survey based)

V-ECUs that contain only some of the "higher software layer parts" of a real ECU (left-hand part of the figure) can be implemented by using the FMI standard. The complete software content of real ECUs (right-hand part of the figure) is often described concerning the AUTOSAR standard. However, because no standard is currently available for intermediate V-ECU levels (middle part of the figure), there is need for action to meet the requirements of these model types and the corresponding simulation use cases. The SmartSE group therefore also evaluated the most relevant and urgent use cases for different V-ECU levels.

In the sections below, this white paper describes a proposal for defining the different abstraction levels of virtual ECUs according to use cases at OEMs, suppliers and IT vendors. In addition, the currently relevant and available standards for (V-)ECUs are described, and their capabilities and limitations regarding their usage for V-ECUs are examined. This is followed by a detailed description of discrepancies and initial (standard-compliant) approaches to a technical solution, based on the possibilities offered by current standards.

This white paper represents a work status from the perspective of the SmartSE V-ECU work package and is intended to provide a basis for further discussion. We will extend this white paper at a later date to incorporate our future work performed by the SmartSE project group. It is also likely that we will publish the results of our V-ECU work package in a future prostep ivip recommendation.

2 Virtual Control Units in the context of Systems Engineering

The complexity of E/E architectures in modern vehicles has been increasing rapidly for years [2]. The need to develop and validate the corresponding functions, software components and networks using simulation methods is increasing accordingly. In the context of automated driving in particular, the need for validation by means of "virtual testing" becomes obvious due to that fact that the extremely large number of tests performed in this context can no longer be executed using only real test drives or Hardware-in-the-Loop systems.

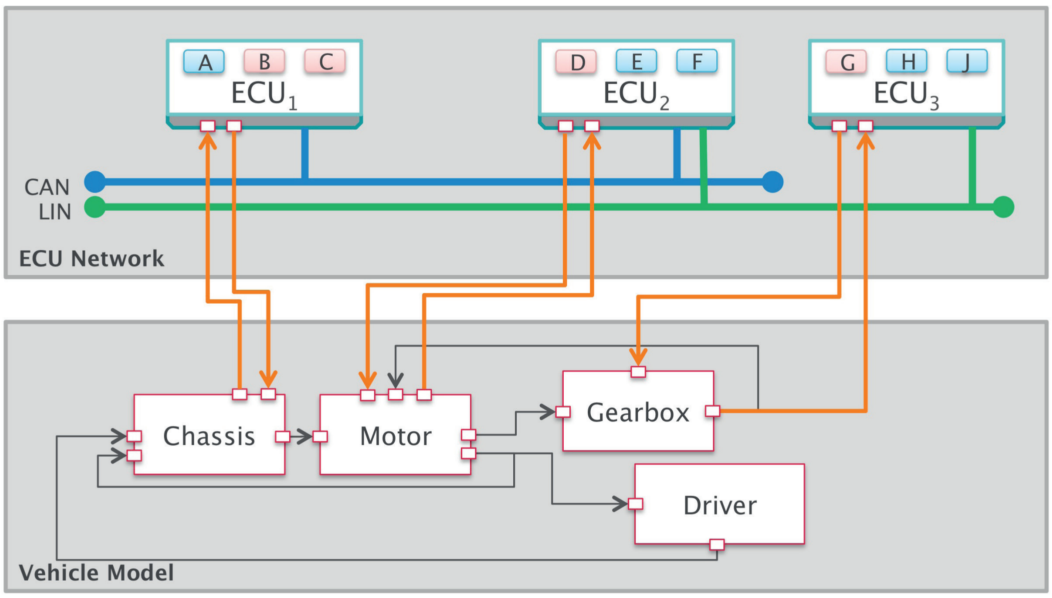


Figure 4 Example of a system model [1]

Figure 4 illustrates an example of a system model with three Electronic Control Units (ECU1 to ECU3) and their inputs and outputs. The ECUs are connected to each other via buses (e.g. CAN, LIN, FlexRay, Ethernet) and to the vehicle model. In addition, control functions within the ECUs are also shown. The different colors indicate that subfunctions on different ECUs have to work together in order to provide the overall control function (e.g. the "blue functions" A, E, F, H,J and the "red functions" B, C, D, G are parts of two control algorithms).

Appropriate exchange formats are needed to implement this type of simulation systems, in which the corresponding models are often provided and integrated by different partners – often even by different companies. Exchange formats like FMI are sufficient for the Vehicle Model parts, but there are currently no adequate exchange formats available for V-ECUs.

Many OEMs now expect their suppliers to provide V-ECUs in ECU projects and to do so at regular intervals in the simplest, quickest and cheapest way possible. Although suppliers receive such requests from a number of different OEMs, they want to create V-ECUs in the same way for all their projects and thus avoid incurring additional costs. They also want to be able to use the V-ECUs themselves internally. Because there is currently no V-ECU standard, OEM specifications are very heterogeneous. This means additional effort and high costs for suppliers, which makes it more difficult for OEMs to get V-ECUs delivered. A V-ECU standard would remedy this situation.

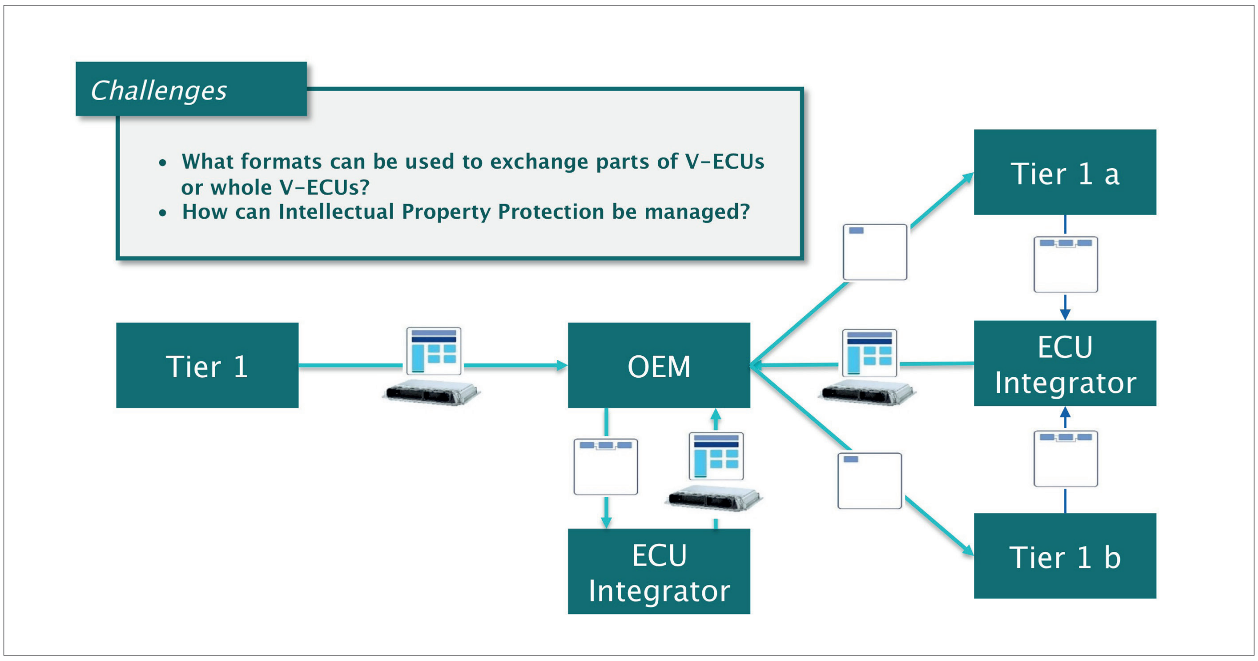

Challenges
What formats can be used to exchange parts of V-ECUs or whole V-ECUs?
How can Intellectual Property Protection be managed?
Tier 1
OEM
Tier 1 a
ECU Integrator
Tier 1 b
ECU Integrator

Figure 5 Data exchange scenarios for generation of V-ECUs

Figure 5 shows that not only complete V-ECUs have to be exchanged between partners like OEMs and Tier1 suppliers. Often, parts of V-ECUs (e.g. individual control functions or parts of control functions) need to be exchanged and integrated into the overall V-ECU by one of the partners. For all of these exchange scenarios corresponding exchange formats need to be standardized to allow partners from different companies to more easily work together and avoid individual exchange solutions. Dependent on the simulation goals, different levels of complexity are necessary. Therefore, this white paper describes in detail what V-ECU levels exist, how these V-ECUs can be generated, and which level is used for which simulation use case.

3 ECU software layers

Normally, the software of an ECU is organized in different so called "layers". Here, we describe the content of the different layers concerning AUTOSAR (Classic Platform) as an example (see Figure 6), but similar layers can also be found in non-AUTOSAR software ECUs.

In AUTOSAR, the "highest layer" comprises the control algorithm itself. This layer is called "Application Layer". The content of this layer is, in most cases, the "system under test" (SUT). If, for example, the control algorithm needs input from an analog channel of the ECU, the "read command" is send through a software layer called "Basic Software". In this layer, different services, e.g. for bus communication, memory services, and input/output drivers, can be found.

Sometimes one part of the application layer needs signals from another part of the application layer within the same ECU. In this case, a direct memory copy can be used for exchange data. But if the signal needed is available on another ECU, inter-ECU bus communication (e.g. via CAN, FlexRay or Ethernet) is required. This means that, depending on the source of the signal needed, the read command could be a single memory copy or a bus communication request, which is handled by the Basic Software layer. Consequently, the "connection" between the application layer and the Basic Software layer must be generated dependent on the arrangement of the software parts within the application layers of different ECUs. The "connection layer" generated in this way is referred to as the Runtime Environment (RTE). Some parts of the Basic Software layer are independent of the underlying microcontroller hardware. Other parts have hardware dependencies. If possible, these hardware dependencies are encapsulated in the so-called Microcontroller Abstraction layer. But sometimes this decoupling from hardware dependencies is not possible. In this case so-called Complex Drivers are used to directly connect the application software with the microcontroller.

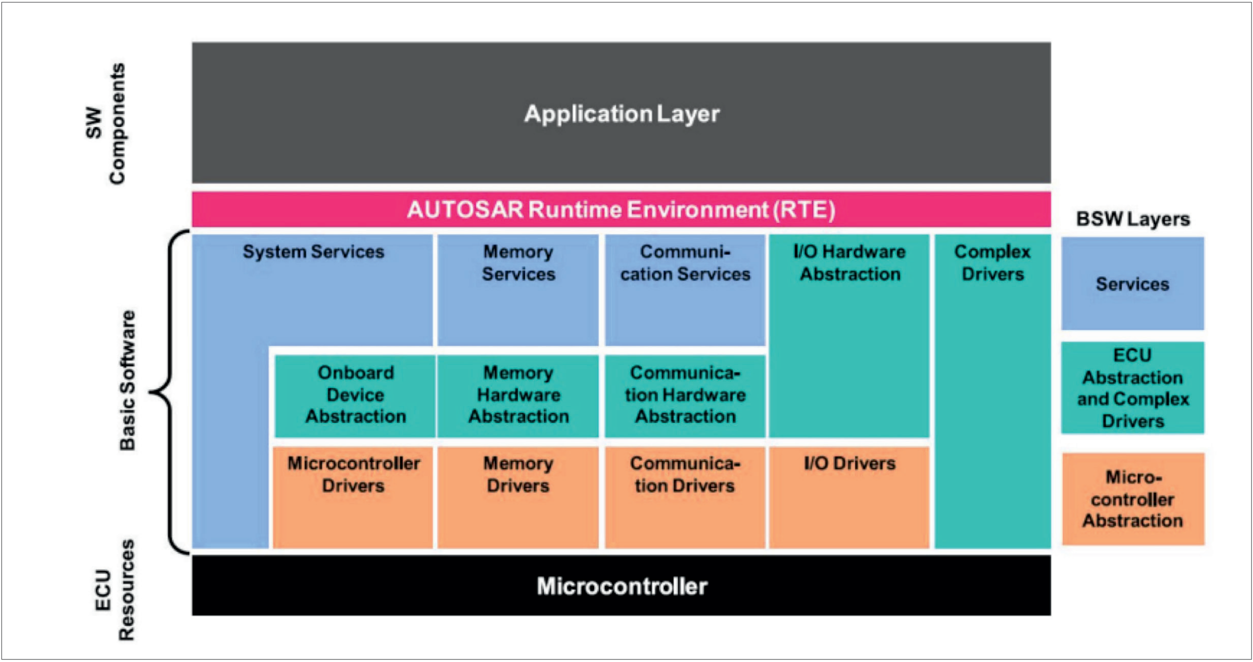


Figure 6 AUTOSAR (Classic Platform) ECU software layers [3]

For the simulation of ECUs, some parts of these software layers must be exchanged, e.g. the hardware drivers for V-ECUs need to be implemented differently compared to the drivers of real ECUs. In most cases, the aim of the simulation is to test the content of the application layer. But because this layer is connected to parts of the Basis Software layer, these software components also often have to be included in the simulation. Dependent on the V-ECU content, different V-ECU levels can be distinguished. The following chapters outline an approach for defining and naming these V-ECU layers.

This description of software layers and the V-ECU layers are based on the AUTOSAR Classic Platform. AUTOSAR [7] has also created what is referred to as the Adaptive Platform [8]. Here, the software architecture is more dynamic and can be configured at runtime. Although this white paper does not include this AUTOSAR Adaptive Platform in detail, some of the software layers can be found there, too. The objective is to ensure that the V-ECU types can also be used to simulate V-ECUs implemented with AUTOSAR Adaptive Platform.

Figure 7 shows a simplified version of the AUTOSAR Basic Software as a reference for the definition of the V-ECU types provided below.

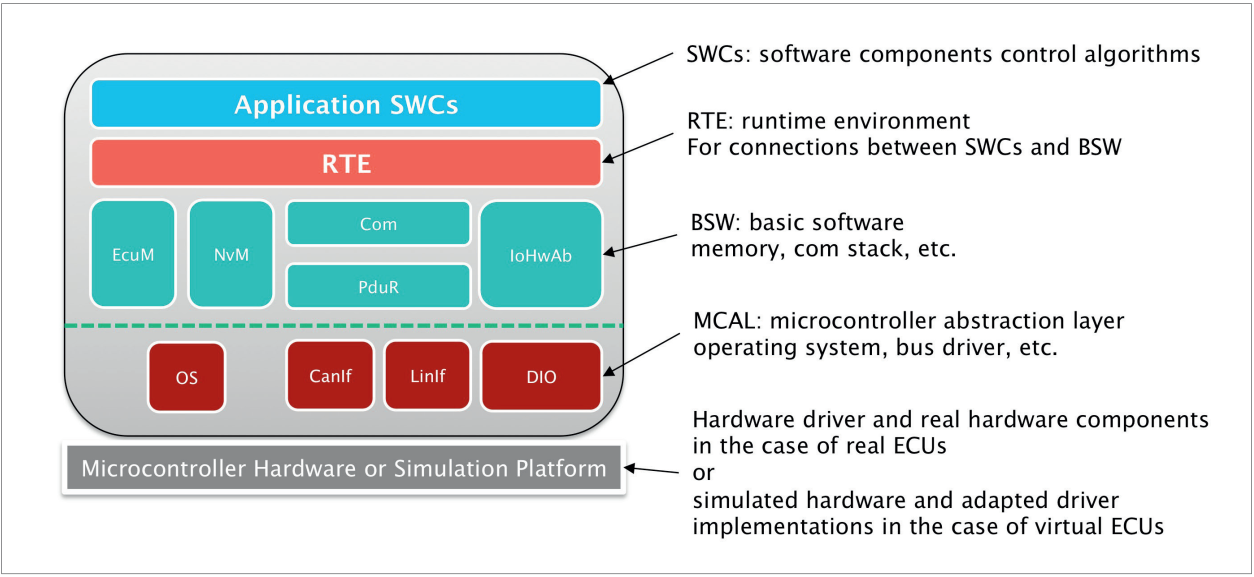

Application SWCs
RTE
EcuM
NvM
Com
PduR
IoHwAb
OS
CanIf
LinIf
DIO
Microcontroller Hardware or Simulation Platform
SWCs: software components control algorithms
RTE: runtime environment
For connections between SWCs and BSW
BSW: basic software
memory, com stack, etc.
MCAL: microcontroller abstraction layer
operating system, bus driver, etc.
Hardware driver and real hardware components
in the case of real ECUs
or
simulated hardware and adapted driver
implementations in the case of virtual ECUs

Figure 7: V-ECUs and the layer model of the AUTOSAR Classic Platform

4 Approach for defining V-ECU levels

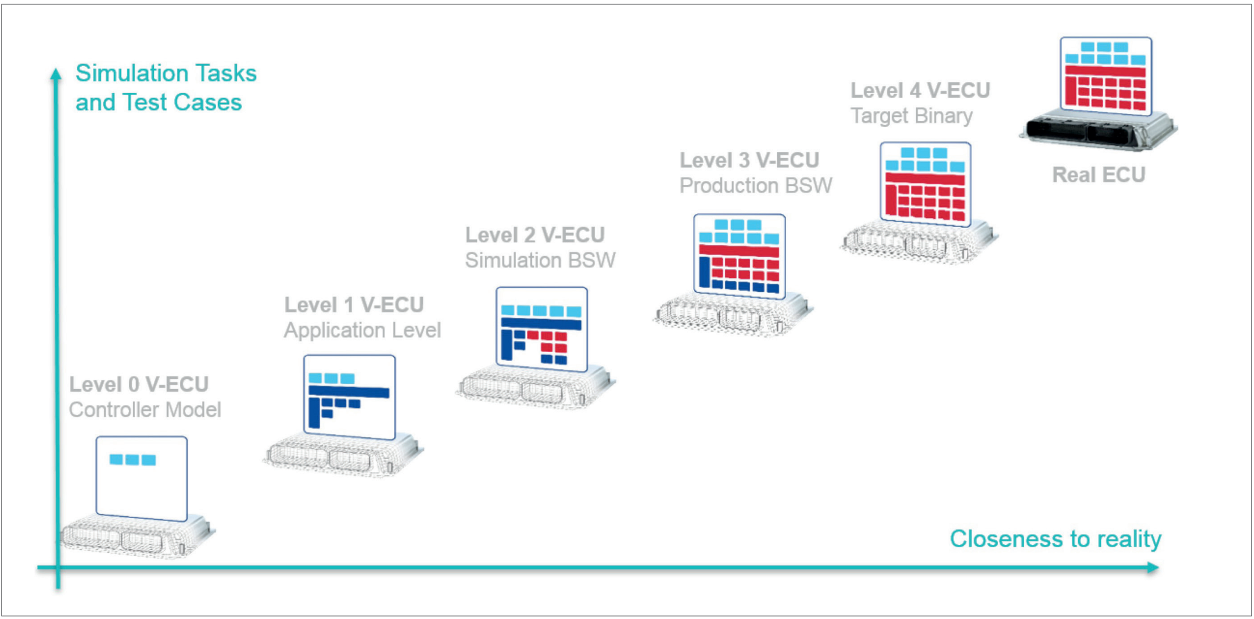


Figure 8 Proposed V-ECU levels

4.1 Definition of the term V-ECU

Until now, no official definition for the term "V-ECU" has been available. It is used in a variety of different contexts, and people often have different things in mind when they use the term. This leads to confusion, and lengthy explanations of how the term is used in a certain context.

Therefore, we want to propose a definition of the term V-ECU, and a definition of the different V-ECU levels:

- A V-ECU is an artifact that contains all the software parts needed to simulate specific aspects of a real ECU.
- The V-ECU software parts consist of C code generated by a controller model or production C code from ECU software development or previously compiled binary code.
- A V-ECU is used for exchange between partners (e.g. OEM and Tier1) and can be executed on simulation platforms like PCs. No real ECU hardware is needed for simulation.
- A V-ECU typically is used for testing the ECU software. In this case, the V-ECU is the "subject under test" (SUT).
- Sometimes V-ECUs are also used in simulation systems for testing the software of other (V-)ECUs. In this case, the V-ECU is needed to provide realistic input to the subject under test and serves as kind of "rest bus simulation".

4.2 Level 0 V-ECU (Controller Model)

A "Level 0 V-ECU" comprises only the controller model itself (e.g. as an MATAB Simulink model) or the C code generated from the controller model (e.g. as an FMU). A Level 0 V-ECU does not include any production code or any part of the basic software layers. This simplest V-ECU type can only be used to test the control algorithm itself. For other tests, e.g. software interface tests, more complex V-ECU types are necessary which are closer to the production code of the real ECU.

4.3 Level 1 V-ECU (Application Level)

A "Level 1 V-ECU" contains production code of the application software. These software components have to be supplemented with functionalities of lower software layers to enable the application software to run as desired during simulation. These typically do not consist of production code but instead are created or generated specifically for the V-ECU. In case of AUTOSAR, these are the "Run-Time Environment" (RTE) and the "Operating System", as well as for example functionality that allows the V-ECU to send and receive data.

The application software parts that the V-ECU contain can be all parts, or a subset of the application software for a real ECU, or it can be application functionality that might later be distributed over multiple ECUs. In any case, it should not include direct hardware accesses, as these direct hardware accesses cannot be simulated on a standard PC.

A Level 1 V-ECU communicates on signal level, it does not use bus or network communication. Level 1 V-ECUs can be used for testing application functionalities distributed over several components (see use case 1 in chapter 9) or ECUs (see use case 3 in chapter 9).

4.4 Level 2 V-ECU (Simulation BSW)

"Level 2 V-ECUs" provide simulated basic software (BSW) functionalities in addition to the content of Level 1 V-ECUs. These included basic software functionalities are not on production software level. They are created specifically for use in the V-ECU for simulation purposes. Examples are a COM stack, NVRAM functionality or diagnostic functionalities.

During simulation, Level 2 V-ECUs can communicate not only on signal level like Level 1 V-ECUs but also at bus or network level (e.g. via simulated CAN or Ethernet communication). Level 2 V-ECUs are used to test the application software. However, the scope of the testing is wider due to the included basic software functionalities. Bus monitoring, connection to rest bus models, diagnostic tests etc. are also possible.

4.5 Level 3 V-ECU (Production BSW)

"Level 3 V-ECUs" include not only production application software but also production basic software (production BSW) or - for tests of the BSW - only the production BSW or parts of the production BSW. Application software and basic software have to be hardware independent. In terms of AUTOSAR, this can be everything above and including the microcode abstraction layer (MCAL layer), the Operating System and those parts of complex device drivers that are hardware independent.

The production basic software can still be supplemented with simulated basic software parts. Also, the application software can be supplemented with functionalities on application level created specifically for the V-ECU. This can be necessary if, for example, some functionalities include hardware dependencies and can therefore not be included in the V-ECU as production code. But, Level 3 V-ECUs typically include as much production code from the real ECU as possible. Level 3 V-ECUs can be used for testing all the hardware independent software of a real ECU (see use case 2 in chapter 9), for testing networks of V-ECUs (see use case 3 in chapter 9), and they can also be used as rest bus models in HIL tests (see use case 5 in chapter 9). Level 3 V-ECUs can also be used to test the BSW itself, e.g. testing the complete or parts of hardware-independent BSW functionality of a real ECU on different test-levels (subsystem-test, stack-test, component-test) . In this case, the application software is not subject under test and can be reduced or replaced to allow tests of the BSW functionalities.

4.6 Level 4 V-ECU (Target Binary)

"Level 4 V-ECU" contains production code compiled for the real target ECU. It is also possible to have hardware dependencies included. This allows all the software layers (e.g. MCAL, OS and complex device drivers) to be included. An instruction set simulation is required to execute this type of V-ECU on a simulation platform.

Level 4 V-ECUs can be used to test the core load of multicore ECUs, for example, (see use case 6 in chapter 9) if the instruction set simulator is sufficiently accurate. For this V-ECU type the identical build process is used for simulation and for flashing the real ECU. No code changes between the V-ECU and the code for the real ECU are required. An instruction set simulator and the model for the corresponding ECU hardware is needed to simulate Level 4 V-ECUs.

	Application Software	Basic Software	Drivers and Hardware Accesses
Level 0 V-ECU Controller Model	Model	–	–
Level 1 V-ECU Application Level	Production Code	–	–
Level 2 V-ECU Simulation BSW	Production Code	For Simulation Purposes	–
Level 3 V-ECU Production BSW	Production Code	Production Code	–
Level 4 V-ECU Target Binary	Binary Code	Binary Code	Binary Code

Figure 9 Overview of V-ECU types and their content3 ECU software layers

5 Overview and short description of use cases for exchanging and simulating V-ECUs

This chapter provides a short description of several use cases for V-ECUs. A more detailed description can be found in chapter 9.

Use case 1: Testing application functionalities
The objective of this use case is to test application software functionalities that are distributed over multiple components. The testing should be performed as early as possible, to provide direct feedback while the components are still developed. Level 1 V-ECUs are normally used for this use case.

Use case 2: Testing all hardware independent ECU software of a single ECU
In this use case, all the production software for an ECU is tested using Level 3 V-ECUs with the aim of ensuring a higher level of software maturity before starting HIL tests using the real ECU. V-ECU tests can be reused later during HIL testing.

Use case 3: Testing a network of V-ECUs
In this use case, the software of multiple, networked ECUs is tested. Level 2 or Level 3 V-ECUs can be used to perform these tests.

Use case 4: Continuous integration and testing
During development of the software, new versions will be continuously integrated in a V-ECU, and automated testing will be performed every time a new version is checked in. A test report is to be given to the developer. Developers can also perform testing manually on their PCs after check-in for debugging purposes or to reproduce errors if testing failed. V-ECUs of all levels are used with continuous integration frameworks.

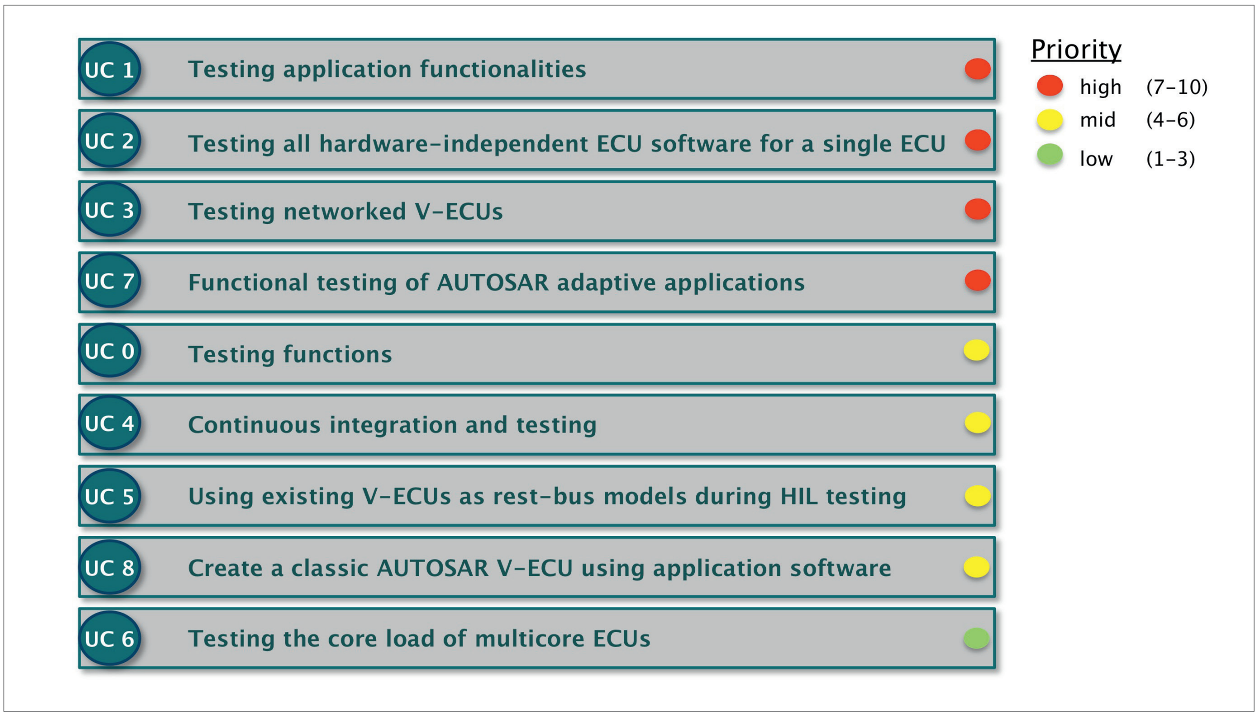
Use case 5: Use existing V-ECUs as rest bus model in HIL tests
When performing HIL tests, existing V-ECUs can be used in place of real ECUs, that are not yet ready, or in place of rest-bus models that represent other ECUs within the overall ECU network. This means that the device under test can be tested earlier, with a more realistic rest bus simulation, and with less effort for creating the bus simulation.

Use case 6: Test of core load of multicore ECUs
The objective here is to test, for example, the computation load of the different cores for a multicore ECU, overall performance, etc. Because all these factors depend on the hardware architecture, a comprehensive instruction set simulator is required.

Use case 7: Functional Test of AUTOSAR Adaptive Applications
The objective here is to perform tests in SIL on a set of functionalities that have been implemented as Adaptive Applications. The tests are to be performed as early as possible in order to provide direct feedback while the components are being developed. In this use case, V-ECUs of different levels are used which have been created in the context of the AUTOSAR Adaptive Platform.

Use case 8: Create a Classic AUTOSAR V-ECU from Application Software
The objective is to create a V-ECU using classic AUTOSAR code that includes Software Components from the Application Software layer. It is intended that the V-ECU be executed on a dedicated simulation platform.

A survey regarding the use cases described above was conducted among the members of the SmartSE project group. The use cases were discussed and assigned a priority by the members. The figure below lists the priorities assigned to the uses cases and indicates that use cases 1, 2, 3, and 7 are of greatest interest. This means that a V-ECU standard should be derived for these use case in particular.



UC 1	Testing application functionalities	🔴
UC 2	Testing all hardware-independent ECU software for a single ECU	🔴
UC 3	Testing networked V-ECUs	🔴
UC 7	Functional testing of AUTOSAR adaptive applications	🔴
UC 0	Testing functions	🟡
UC 4	Continuous integration and testing	🟡
UC 5	Using existing V-ECUs as rest-bus models during HIL testing	🟡
UC 8	Create a classic AUTOSAR V-ECU using application software	🟡
UC 6	Testing the core load of multicore ECUs	🟢

Priority

🔴	high	(7–10)
🟡	mid	(4–6)
🟢	low	(1–3)

Figure 10 Prioritized V-ECU Use Cases

6 Existing Standards for V-ECU usage. Opportunities, limits and challenges

The following two chapters address possible solutions for a V-ECU standard with focus placed on V-ECU levels 1 to 3. It is already possible to exchange level 0 V-ECUs (controller models) via FMI. Level 4 V-ECUs for instruction set simulation are technically very different from level 1, level 2 and level 3 V-ECUs. The objective of the following chapters is to describe initial ideas for a standard for the types of V-ECUs, which are based on production source code. The standard must be able to support all three levels of V-ECUs.

6.1 FMI

The basic idea behind the Functional Mock-up Interface (FMI) is simplifying the exchange of models between OEMs and suppliers by means of a standard and defining a standard for co-simulation of the models. The FMI for co-simulation describes an extensive, standardized C-API, which is represented in its concrete instance by a Functional Mock-up Unit (FMU) [4].

Basically, an FMU is a compressed file that contains an XML description and source code or precompiled binaries. Source code enables easy adaptation of the FMU and allows it to be delivered independent of the simulation platform. Pre-compiled binaries, on the other hand, offer IP protection and can be provided for different platforms (32-bit/64-bit, Linux/Windows) within the framework of the standard.

In addition to the step size for execution by a simulator, the XML description also specifies inputs/outputs and parameters. The current version of the FMI standard (version 2.0) is limited to mapping scalar data types such as Boolean, Integer, String and Real. Without a user-specific and non-standardized implementation, FMI 2.0 does not provide an efficient way to exchange larger amounts of data. The representation of complex structures or even large binary data is not covered by FMI 2.0. Explored approaches to exchange corresponding address pointers with the previous means of the standard via the standard data types can only provide a limited remedy [4]. By explicitly accessing the memory of the execution platform, the ideas of the standard are circumvented and, in particular, platform independence may be violated. When using complex scheduling such as multiple cycle times or events, the limitations of the standard also necessitate an application-specific solution within the FMU implemented for co-simulation.

The limitations of the current version FMI 2.0 have already been identified and addressed in part by the upcoming version 3.0. Nevertheless, two key points involving the representation of complex virtual ECUs using FMI 3.0 remain open: addressing bus communication such as CAN, FlexRay or Ethernet, and the fact that there is currently no concrete release date for version 3.0 [6].

This document does not consider the latest developments of the FMI 3.0 standard, as this version is not yet officially released. Some parts of this document may not reflect the view of the Modelica Association FMI project group, but the authors of this document are in close contact to the FMI project group to clarify, whether FMI 3.0 has to be extended to be used for all types of V-ECU levels. The current specification of the unreleased FMI 3.0 Standard can be found in the dedicated Git-Hub [9].

6.2 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry. AUTOSAR aims to improve complexity management of integrated E/E architectures through increased reuse and exchangeability of software modules between OEMs and suppliers. AUTOSAR standardizes the software architecture of electronic control units (ECUs) and is an active committee with regular releases and defined processes.

Compared to the FMI, AUTOSAR does not standardize a container that can be executed directly in a simulator, but rather an ECU software architecture. In principle, this can also be used to define interfaces between the code contained in the V-ECU and the simulation platform. This is possible in particular due to the fact that AUTOSAR defines a clear interface between the hardware-dependent and the hardware-independent part of the ECU code. In the AUTOSAR software architecture, the operating system (OS) and the microcontroller abstraction layer (MCAL) is the natural interface to the simulation of the ECU hardware and communication. The OS and MCAL modules could be used as a basis for a V-ECU standard by defining additional interfaces specifically for a simulator and a container format.

7 Technical Solution Approaches

This chapter describes a possible technical approach for developing a V-ECU standard. Section 7.1 first lists requirements for a technical V-ECU standard. These are based on the previous chapters and in particular on the use cases. This is followed by the description of a possible approach to a standard. This approach includes the technical interface between V-ECU and simulator (section 7.2), and a concept for a container format for exchanging V-ECUs as artifacts rather than multiple files (section 7.3).

7.1 Requirements for a V-ECU standard

A V-ECU standard must meet a number of requirements if it is to support the above-mentioned use cases and thus also gain wide acceptance. These requirements are described in this section.

Requirement 1: It must be possible to exchange V-ECUs between tools from different vendors, i.e. the tools must be able to understand V-ECUs from other vendors.
Reason: It should be possible to exchange V-ECUs between OEMs and suppliers without have to use the same tool vendor.

Requirement 2: All V-ECUs on the different levels must be supported. Communication between the V-ECU and the outside world must be possible via buses, network and signals.
Reason: Important use cases exist for all levels of V-ECUs.

Requirement 3: It must be possible to exchange V-ECUs with IP-protected parts, e.g. by compiling the C code before the V-ECUs are exchanged.
Reason: Exchanging V-ECUs between different partners is often only possible with IP protection.

Requirement 4: It must be possible to apply the standard to different simulation platforms (e.g. real-time platforms like HIL systems or non-real-time SIL platforms (e.g. standard PCs or cloud systems).
Reason: There are use cases involving the use of V-ECUs on different simulation platforms.

Requirement 5: It must be possible to run a concrete V-ECU on different simulators. In the case of IP-protection, this means that several compilates may be included.
Reason: A V-ECU from one partner that is provided to another partner can be used for different use cases on different platforms.

Requirement 6: It must be possible to use the standard for both AUTOSAR (Classic and Adaptive) and non-AUTOSAR ECUs.
Reason: The standard should be valid for all ECU projects.

Requirement 7: It must be possible to integrate an XCP service in the V-ECU.
Reason: It must be possible to access measurements and calibration during the simulation.

Requirement 8: The standard must describe how bus and network simulation can be implemented. The V-ECUs should have special "bus ports" to describe the connection between V-ECUs via bus or network.
Reason: It must be possible to simulate a complete V-ECU network and communication between multiple V-ECUs has to be done in a way that is as similar as possible to the behavior of the real bus/network.

Requirement 9: The standardization committee must ensure that clear statements regarding the roadmap are made and that the standardization of V-ECUs can be implemented in the short-term.
Reason: There is enormous time pressure when it comes to implementing the standardized exchange of V-ECUs.

7.2 Standardized API for V-ECU Code

This section describes possible approaches to a standardized API for V-ECU implementation. AUTOSAR and FMI are viewed as possible standards for V-ECUs or as the basis for a V-ECU standard.

In the case of AUTOSAR, the Microcontroller Abstraction Layer (MCAL) could be used as the part of the software with direct hardware accesses, and the upper layers of the AUTOSAR architecture could be used as the API for V-ECU implementation. In addition, interfaces for signal-based communication and for controlling the simulation would have to be created. One disadvantage of this approach is that this interface is relatively complex and is only used by AUTOSAR ECUs. Although the creation of V-ECUs for AUTOSAR ECUs would therefore be relatively easy, the creation of V-ECUs for non-AUTOSAR V-ECUs would be complex, since all interfaces would have to be mapped to the AUTOSAR interfaces.

FMI, on the other hand, offers a much simpler interface. A simpler interface has the advantage that it makes it easier for simulator vendors to support the standard, which should lead to a higher level of acceptance. Generally speaking, FMI is a good approach to creating a V-ECU standard, in particular with the feature set that will be offered by FMI 3.0. FMUs are easy to create using the standardized and easy-to-use C API. But FMI does not provide a comprehensive solution for implementing level 2 and level 3 V-ECUs due to the limitations when it comes to exchanging complex data, e.g. bus and network communication. FMI is therefore a good choice for representing level 0 and level 1 V-ECUs. But due to its limitations when it comes to representing complex virtual ECUs, especially with basic software components and realistic bus behavior, the FMI 3.0 standard is not comprehensive enough to be used directly as V-ECU standard.

In view of the relevant use cases and derived requirements, a bus interface that allows for realistic bus simulation between V-ECUs from different parties is essential. An extension of or supplement to the FMI standard could therefore be one option for implementing a V-ECU standard. It would however need to define a bus interface, for which technical concepts would have to be developed.

A standard could therefore define a V-ECU API based on FMI 3.0 (or a later FMI version). This approach involves a number of issues that need to be taken into consideration. This FMI version would have to be released in time. The definition of the V-ECU API would also have to be pursued in a committee that would work on it together with appropriate experts and could guarantee timely implementation. Finally, the definition of the V-ECU standard may reveal additional requirements that would have to be implemented in the FMI standard in a timely manner.

7.3 Container Format for V-ECUs

An API for the V-ECU is one part of a standard. The other part is a container format that can be used to exchange the V-ECUs, and which can be understood by supporting tools. The container format should also be standardized by the same committee that standardizes the API interface. This section lists a few aspects that need to be taken into consideration.

The container format should include the source code for the V-ECU, either as plain code or, for IP protection reasons, in a form compiled for a dedicated simulator platform. It should be possible to have several compilations of the V-ECU code in one container, so that the V-ECU can be executed on several platforms. The container should also contain an A2L file for the measurement and calibration of variables during the simulation. Additional information, e.g. meta information or initial data for NvRAM, might also be needed.

8 Roadmap and next steps

Figure 11 shows the 2019/2020 roadmap for the V-ECU work package. As far as the short- and medium-term outlook is concerned, the next step is to incorporate feedback and contributions to the discussion arising from this document. The technical requirements for standardization will be derived once a detailed definition has been drawn up and the concept has been finalized. In this context, it is also necessary to discuss which requirements are to be addressed with regard to AUTOSAR / FMI and if another standardization organization should be involved. The proposal formulated with regard to procedures and content will also be part of an upcoming SmartSE prostep ivip recommendation.

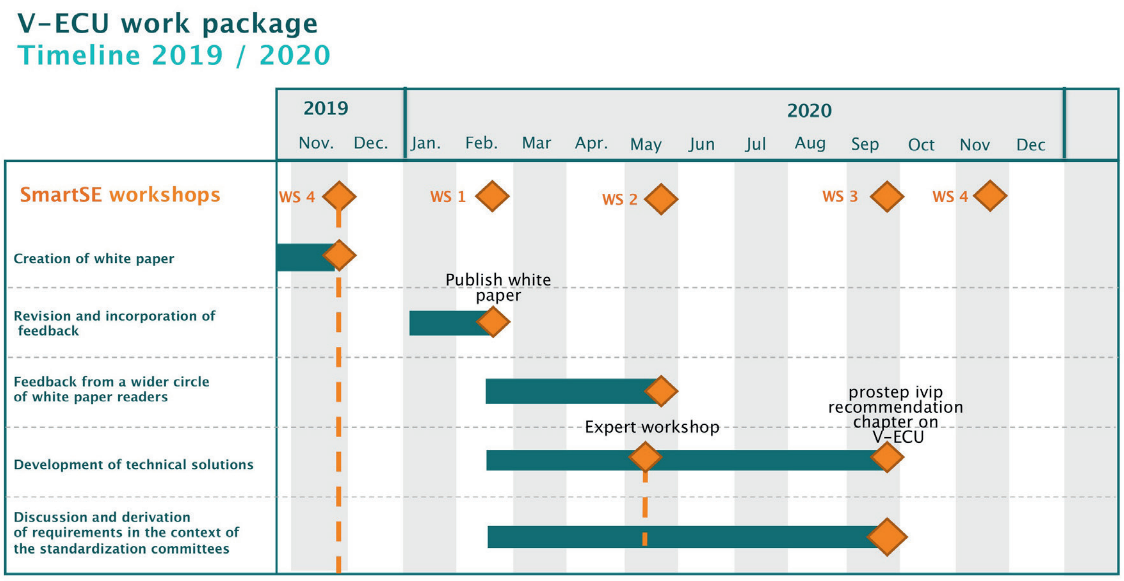


Figure 11 Roadmap Work Package V-ECU

9 Appendix – Use Case Description

9.1 Use Case 1: Testing application functionalities

An API for the V-ECU is one part of a standard. The other part is a container format that can be used to exchange the V-ECUs, and which can be understood by supporting tools. The container format should also be standardized by the same committee that standardizes the API interface. This section lists a few aspects that need to be taken into consideration.

Goal
The objective is to test application software functionalities that are distributed over multiple components. The testing should be performed as early as possible to provide direct feedback while the components are still being developed.
V-ECU type used: level 1 V-ECU

Preconditions
- The functionalities have been implemented
- The software is platform-independent or has been prepared for execution using a PC as the simulation platform
- One or more V-ECUs that include the functionalities to be tested have been prepared
- Rest-bus and environment models are available

Process / Execution
- Connect the V-ECUs and integrate them with rest bus and environment models
- Create test scripts, layouts, etc.
- Start simulation
- Calibrate parameters
- Collect and report results

Results
- The functionalities are tested in SIL
- Test results are available

9.2 Use Case 2: Testing all hardware-independent ECU software of a single ECU

Goal
The objective is to test the complete ECU software to ensure a higher level of software maturity before starting HIL testing. Tests should be reusable in HIL tests as far as possible.
Used V-ECU type: level 3 V-ECU

Preconditions
- The production ECU code is available as Classic AUTOSAR or Non-AUTOSAR code
- The software has been prepared for executing it on the PC as simulation platform
- All hardware-dependencies in the code have been eliminated
- A V-ECU containing the complete hardware-independent ECU code has been prepared
- Rest bus and environment models are available

Process / Execution
- Integrate the V-ECU with rest bus and environment models
- Create test scripts, layouts, etc.
- Start simulation
- Calibrate parameters
- Collect and report results

Results

- The V-ECU and therefore the complete ECU software is tested in SIL
- Test results are available
- Tests, layouts can be reused later in HIL simulation

9.3 Use Case 3: Testing networked V-ECUs

Goal

The objective is to test the software of multiple, networked ECUs in SIL and the network's communication matrix.
Used V-ECU type: level 2 V-ECU, level 3 V-ECU

Preconditions

- All V-ECUs have been prepared
- The V-ECUs communicate on the same level (bus level vs. signal level)
- Additional models, e.g. rest bus and environment models, are available

Process / Execution

- Connect the V-ECUs with each other and rest bus and environment models
- Create test scripts, layouts, etc.
- Start simulation
- Collect and report results

Results

- The V-ECUs are tested in SIL
- Test results are available
- Tests, layouts can be reused later in HIL simulation

9.4 Use Case 4: Continuous integration and testing

Goal

During development of the software, new versions will be continuously integrated in a V-ECU and automated testing will be performed every time a new version is checked in. A test report is to be given to the developer. Developers can also perform testing manually on their PCs after check-in for debugging purposes or to reproduce errors if testing failed.
Used V-ECU type: level 1 V-ECU, level 2 V-ECU, level 3 V-ECU

Preconditions

- A new software version is available for the integration
- A V-ECU for the current software status has been prepared
- Plant models etc. have been prepared for the respective integration level
- Tests have been automated
- A continuous integration process is available

Process / Execution

- Check in the new software version for one software part
- Wait for continuous integration process to update V-ECU and execute automated tests
- Read test results
- In case of failed tests: reproduce tests in the same environment on the PC, debugging
- Fix errors and check in again, until all tests are passed

Results

- The new software version integrated with the complete functionality is tested in SIL
- Bugs that have been found are fixed

9.5 Use Case 5: Using existing V-ECUs as rest-bus model during HIL testing

Goal

When performing HIL testing, existing V-ECUs can be used in place of real ECUs that are not yet ready or instead rest-bus models that represent other ECUs within the overall ECU network. This means that the ECU which is device-under-test can be tested earlier and with more realistic rest-bus simulation which, in addition, can be created with less effort.
Used V-ECU type: level 1 V-ECU, level 2 V-ECU, level 3 V-ECU

Preconditions

- V-ECUs exist for the ECUs and rest bus models to be replaced in the test
- The V-ECUs can be connected to the ECU on bus or network level
- The device under test ECU and a HIL simulator are available

Process / Execution

- Connect V-ECUs with the ECU
- Execute tests on the HIL Simulator

Results

- The ECU is tested on a HIL without the need for ECU prototypes of connected ECUs and rest bus models

9.6 Use Case 6: Testing the core load of multicore ECUs

Goal

The objective is to test, for example, the computation load of the different cores for a multicore ECU, overall performance, etc. Because all these factors are dependent on the hardware architecture, a comprehensive instruction set simulator is required.
Used V-ECU type: level 4 V-ECU

Preconditions

- Level 4 V-ECU is available
- Instruction set simulator is available for hardware emulation

Process / Execution

- Connect the V-ECUs and integrate them with rest bus and environment models
- Create test scripts, layouts, etc.
- Start simulation
- Calibrate parameters
- Collect and report results

Results

- Performance of multicore ECU architecture is tested

9.7 Use Case 7: Functional testing of AUTOSAR Adaptive Applications

Goal

The objective here is to perform functional testing in SIL on a set of functionalities that have been implemented as Adaptive Applications. The tests are to be performed as early as possible in order to provide direct feedback while the components are being developed.
Used V-ECU type: level 3 or level 4 V-ECU created according to AUTOSAR Adaptive Platform

Preconditions

- The Adaptive Applications are available
- One or more V-ECUs containing the functionalities to be tested have been prepared
- If applicable, rest bus and environment models are available

Process / Execution

- Connect the V-ECUs and integrate them with rest bus and environment models
- Create test scripts, layouts, etc.
- Start simulation
- Collect and report results

Results

- The functionalities are tested in SIL
- Test results are available

9.8 Use Case 8: Create a Classic AUTOSAR V-ECU from Application Software components

Goal

The objective is to create a V-ECU using classic AUTOSAR code that includes Application Software components. It is intended that the V-ECU be executed on a dedicated simulation platform.
Used V-ECU type: level 1 V-ECU

Preconditions

- Arxml files for the software components are available
- Implementations of the software components are available
- The implementations are prepared in a way that they can be executed on the desired simulation platform

Process / Execution

- Generate an RTE and Operating System for the software components
- Make the RTE signals available for external communication
- Create ports for the V-ECU to communicate with other simulation participants
- Wrap everything in a V-ECU format that is supported by the desired simulation platform

Results

- A level 1V-ECU is available and can be executed on the desired simulation platform
- The V-ECU can be used for simulations of systems (with other V-ECUs and environment model)

10 References

[1] Rühl, M. (dSPACE GmbH); Heinkel, H.-M. (Robert Bosch GmbH): Systems Engineering – Testen autonomer Fahrfunktionen mit virtuellen Steuergeräten, Vortrag ProSTEP Symposium 2018

[2] G. Reichart und J. Bielefeld. Einflusse von Fahrerassistenzsystemen auf die Systemarchitektur im Kraftfahrzeug. In Handbuch Fahrerassistenzsysteme, Seiten 84–92. Springer, 2009.

[3] C. Acar, https://medium.com/@can.acar/autosar-fundamentals-what-is-autosar-part-1-ac6198c4b075, Feb. 2019, aufgerufen am 14.10.2019

[4] FMI Standard, https://fmi-standard.org, Nov. 2019, aufgerufen am 13.11.2019

[5] OSI & FMI, https://github.com/OpenSimulationInterface/, Nov. 2019, aufgerufen am 13.11.2019

[6] FMU 3.0 New Features, https://fmi-standard.org/faq/#what-will-be-the-new-features-of-fmi-30, Nov. 2019, aufgerufen am 13.11.2019

[7] AUTOSAR, https://www.autosar.org/, Nov. 2019, aufgerufen am 26.11.2019

[8] AUTOSAR Adaptive Platform, https://www.autosar.org/standards/adaptive-platform/, Nov. 2019, aufgerufen am 26.11.2019

[9] github-Repository, https://github.com/modelica/fmi-standard/

prostep ivip association

Dolivostraße 11
64293 Darmstadt
Germany

Phone +49-6151-9287336
Fax +49-6151-9287326

psev@prostep.com
www.prostep.org

ISBN 978-3-9820795-1-6
Version 1.0, 2020-3
PSI 31