Testing Automotive Embedded Systems under X-in-the-Loop Setups

Invited Paper

Ghizlane Tibba[1], Christoph Malz[1], Christoph Stoermer[1], Natarajan Nagarajan[1], Licong Zhang[2], and Samarjit Chakraborty[2]

[1]ETAS GmbH, Stuttgart Germany

[2]Institute for Real-Time Computer Systems, Technical University of Munich

[1]{Ghizlane.Tibba, Christoph.Malz, Christoph.Stoermer, Natarajan.Nagarajan}@etas.com

[2]{licong.zhang,samarjit}@tum.de

ABSTRACT

The development of automotive electronics and software systems is often associated with high costs due to their multi-domain nature (including control engineering, electronics, hydraulics, mechanics, etc). The involvement of these different disciplines makes it difficult for control engineers to test their controllers with them being integrated in the whole system, in early development phases. By introducing the "XiL approach", ETAS wants to leverage virtualization techniques in order to bring embedded systems to the desk of every developer. The XiL approach implies fast development cycles due to easy integration of software, vehicle and plant components on the PC. XiL strives for seamless transition between X-in-the-loop setups, with X representing any control model (M), software (S), or hardware (H) under test. This paper will outline this methodology with an engine management system example.

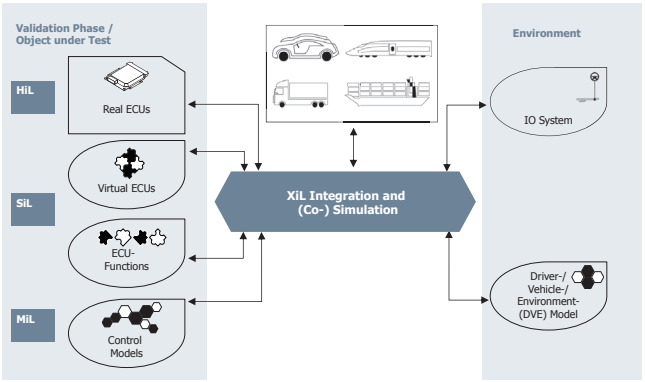

Figure 1: XiL in embedded software and system development.

1. INTRODUCTION

As both the functional algorithms and the underlying architecture of the automotive embedded systems are becoming increasingly complex, engineers require constant testing during the whole development process, in order to design and develop efficient and reliable software. It would be advantageous for the engineers to be able to test their algorithms from an early development phase and have a consistent testing framework throughout the whole development process, for example, from the model level to the software level and eventually to the hardware level. However, due to the multi-domain nature of the development of automotive electronics and software systems (including control engineering, electronics, hydraulics, mechanics, etc.), it is difficult for the engineers to test their electronics and software in early design phases. This is especially true for embedded control software, since it is difficult to test with the physical plant until later stages of the development process. An example is the powertrain system, where testing of the engine controller is often an overwhelming undertaking due to the consideration of interactions between all sub-components (e.g., air, fuel and ignition system). Therefore, simulation-based testing is an essential method to enable control engineers to test their controllers in early stages of the development. Whereas hardware-in-the-loop (HiL) testing has been the main approach for simulation-focused testing in the automotive domain a few years ago, model-in-the-loop (MiL) and software-in-the-loop (SiL) have also become frequently used today. The X-in-the-loop (XiL) approaches use virtualization of the physical plants and the embedded system architectures to allow control engineers to test the controllers on the model, software and hardware level respectively and thus achieve faster development cycles.

The main challenges for an integrated XiL approach are: components integration and the transition between XiL lev-

ICCAD '16, November 07-10, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2966986.2980076

els. First of all, simulation models might be modeled with different tools such as MATLAB/Simulink [1] or ASCET [2]. To integrate these components in a simulation framework is not a trivial task. In an integrated approach, the simulation framework needs to handle different models of computation (MoC) (e.g., discrete-event, discrete-time, continuous-time, etc.). Each model of computation has a specific mechanism for time progression and event handling [25]. The integration platform must be able to handle system compositions that use different models of computation while respecting the time synchronization between them. The platform must also enable integration of models by means of capturing the communication (with any translation that might be needed) that occurs between them. In addition, controllers on different levels of XiL simulation contain different levels of implementation details. In an integrated approach, the simulation framework needs to handle these different levels of implementation details from MiL to SiL and to HiL and provide a seamless transition throughout these levels, which is a challenging task. In addition, one of the main aims of XiL simulation is to facilitate testing. Therefore efficient support of test case reuse between different XiL levels, different products in the same family and between product generations is an important issue, which also needs to be addressed. Towards addressing the first problem, efforts to standardize the exchange of simulation models and enable integration in co-simulation scenarios were initiated by the ITEA2 MODELISAR project [3]. The main result here was the development of the functional mockup interface (FMI) standard, which enables the exchange of dynamic simulation models between different simulation tools. However, this standard only defines the simulation procedure for each FMI simulation model within one simulation step, but does not specify the details of how to handle the different models (e.g., how the models are synchronized and executed). Neither does the standard address the issue of test case reuse.

In this paper, we present a new XiL approach (Figure 1) of ETAS based on the FMI standard that addresses the aforementioned challenges. This approach and its tool support is based on the FMI standard and provide a simulation orchestration mechanism that can efficiently handle the combination of different models of computation. Furthermore, our approach uses the black box testing method to facilitate the test case reuse. This approach enhances efficiency and quality in design, integration, calibration and testing of the development of automotive embedded systems. It offers the control engineers the option to have his whole test environment on a single PC and provides a seamless transition between the different XiL levels. Furthermore, it facilitates test case reuse between XiL levels, products and product generations. The experimental results have illustrated this seamless transition as well as the accuracy and performance of this integrated approach and its tool support.

The rest of this paper is organized as follows. Section 2 reviews the related work and Section 3 explains the building blocks of the functional mockup standard and the definitions of the XiL levels. Section 4 describes the approach we developed and implemented to support the seamless XiL. The experimental results are then shown in Section 5, where we use an engine management system to illustrate the performance of our approach. In Section 6, we present the concluding remarks and possible future work.

2. RELATED WORK

The first category of related work concerns the development of the FMI standard. Since the first release of the standard, efforts have been made in order to implement and test the functional mock-up units (entities implementing the standard). In terms of building new workflows for simulation and verification of systems under development, [4] proposes a simulation environment for continuous-time systems based on FMI, but does not address the discrete or the hybrid systems discussed in our approach. [5] explains how to achieve deterministic execution of functional mockup units (FMU) for co-simulation. However, it does not target the functional mockup units for model exchange which we consider for discrete-event simulation in our approach. [7] discusses technical issues and the implementation of a generic Modelica [8] interface to support FMI import in Modelica simulators. [6] compares the simulation results between models without FMUs and FMUs of a vehicle power network. [10] describes how to create FMUs using SystemC and SystemC-AMS from integrated electric and electronic system models together as well as their corresponding simulation engines. [9] discusses limitations for the usage of FMI in ECU software and proposes an extension of the standard to fulfill these requirements. But this work do not address the complete workflow on how to leverage FMI standard to support the model exchange of hybrid models as well as their co-simulation and test case reuse potential through XiL levels presented in our approach.

Secondly, in the community of cyber-physical systems, the research focus of control and platform co-design has recently been drawing increasingly more attention. The aforementioned approaches focus more on developing an integrated XiL simulation framework to allow the engineers convenient testing of their designs, without addressing the actual design problem. In contrast, the control/platform co-design methods concentrate on synthesizing the design parameters for both controllers and the architecture from a joint specification. One example of such approaches is [11], where the design of controllers and the schedules is integrated into a holistic framework and the sampling period of the control applications and the schedules can be synthesized while optimizing the control performance. Furthermore, [12] have addressed a similar problem while integrating the design of controller gains in addition. Here both the controllers and the platform parameters can be optimized according to both the overall control performance and the bus resource utilization and trade-off options between the both are offered. Overall, the control/platform co-design methods target at synthesizing both control and platform parameters according to the specification, while taking the implementation details into consideration. However, one main challenge amongst others facing these approaches is the lack of convenient tool chain support. One important aspect of the tool support is the simulation on different design and implementation levels. Although these methods employ the correct-by-design approach, additional simulation testing would still be greatly helpful for the validation if these approaches were to be ap-

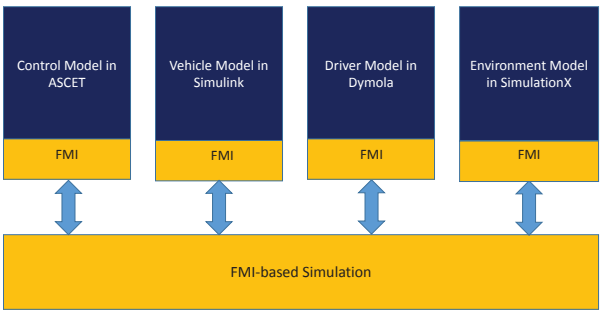

Figure 2: System integration through FMI.

plied for industrial products. In addition, simulation results might help the designers to evaluate and choose different design options offered, e.g., in [12]. In this sense, the approach proposed in this paper and the co-design approaches are complementary to each other, i.e., the co-design approaches can help designers to automatically synthesize control and architecture parameters for control software from a joint specification, but need further evaluation and validation instrument and the XiL approach presented in this paper, although not addressing the design problem, can be utilized to provide the evaluation and validation option.

3. BACKGROUND

3.1 Functional Mockup Interface Standard

As different modeling/simulation tools are in use during the described development process, a prerequisite for setting up heterogeneous simulation systems is the possibility to integrate simulation models that were created with different tools, such as ASCET, MATLAB/Simulink. In order to achieve exchangeability of simulation models, the FMI standard [3] was developed. FMI is a tool-independent standard for the exchange of dynamic models and for co-simulation. The development of the standard was initiated and organized within the ITEA2 project MODELISAR [13]. Its primary goal is to support the exchange of simulation models between suppliers and original equipment manufacturers (OEMs) even if a large variety of different tools are used. As of today, development of the standard continues through the participation of 16 companies and research institutes [3].

The FMI standard defines an open interface to be implemented by functional mock-up units (FMUs). Thus a control model that is exported as FMU from a tool like ASCET can be simulated in a FMI-compliant simulator with FMUs from other tools, as shown in Figure 2. The FMUs by themselves are passible objects and a master algorithm (MA) is needed to schedule and execute them. The MA then coordinates the execution of a number of interconnected FMUs in simulation runs [14].

An FMU is a software library distributed in a zip file with FMU file extension containing (i) a model description file, (ii) model equations and (iii) optional resource files. Firstly, the model description file contains information about the model, for example variable definitions type, unit, description, etc., as well as other more general model information, such as model name, generation tool, and FMI version. Secondly, a model can be described using ordinary differential equations, algebraic relations, and discrete equations including time, state, and step events. These equations can in turn be represented by a small set of C functions. The C code is then distributed in the FMU in source and/or binary form, and one FMU can contain binaries for more than one platform and/or platform versions. Finally, there are other optional files that might be included in the FMU, such as documentation files (HTML), model icon (bitmap file), maps and tables, and other libraries or DLLs that are used in the model. Although the FMI standard is designed to enable the exchange of models, it encourages the inner structure of the models to remain hidden. It allows the use of binaries instead of readable C code and hiding of the inner variables in the xml model definition [15].

FMI describes a standardized access to model equations, enabling a mixed usage of continuous-time, discrete-time and discrete-event mechanisms in FMI for model exchange, all of which could appear in a mechatronic system. Functions in the FMI can be categorized in the following categories:

- **Initialization and instantiation functions:** These are used to load the component, supply the parameters and allocate memory.
- **Progress functions:** Functions to move forward the time.
- **Getter and setter functions:** Getter functions are used to read the output values and their derivatives. Setter functions can be used to set input values and their derivatives for interpolation.
- **Termination functions:** These are used to unload the component and free the memory.

FMI further offers two variants, namely (i) FMI for model exchange and (ii) FMI for co-simulation. In FMI for model exchange, the model is described by differential, algebraic and discrete equations with time, state and step event. These equations are then to be solved using a standardized C interface. The solver must be provided by the host tool. On the other hand, FMI for co-simulation provides a means to utilize models using an API, in a form where the slave (i.e., FMU) acts as a black box to the master. It can react to inputs and give outputs at discrete time steps. While using an FMU conforming to FMI for co-simulation, one does not need to know which integration method is actually applied to solve the model. Parameters to this black box can be set in the initialization phase and cannot be changed afterwards, while the inputs can be changed between discrete time steps [16].

3.2 XiL levels

In the context of model-based development in the automotive domain, the testing process of an ECU can be divided into subsequent test-levels as follows.

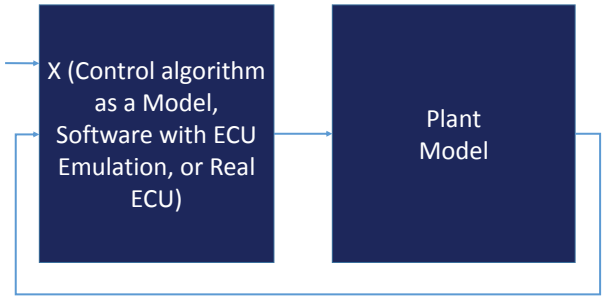

Figure 3: Generic X-in-the-loop setup.

Model-in-the-loop: Under the MiL testbench, the control model is integrated with a model of the plant, which captures the dynamics. Both the plant and the control models are simulated together to verify the functionality without any hardware components [17, 18] and without suffering from special limitations of the target environment [19].

Software-in-the-loop: There are several understanding of the SiL testbench. In the work of [17], SiL is described as the control software being available and integrated together with the plant model without the need of any hardware or real-time environment. This is also referred to as MiL-to-SiL by [20]. The SiL simulation described in [21] has parts of the model existing in a standard simulation tool such as Simulink and the rest is compiled code. In both cases, the C code is compiled for the host PC instead of the target microcontroller. This may introduce difference in precision due to different data types (saturation and overflow effects), as well as other problems due to resource limitations on the microcontroller (memory and processing power) [22]. In [20], SiL describes the control software running on an emulated ECU and is simulated against the plant model without a need of any hardware.

Hardware-in-the-loop: Under HiL setup, the control software runs on the target ECU. The plant is still simulated but running on a real-time PC. The interaction between the ECU and the plant is conducted via the digital and analog electrical connectors of the ECU. In a higher abstraction level, and as described in Figure 3, MiL, SiL or HiL have the same overall architecture, where the X under test is either model, software or hardware.

4. XIL APPROACH

For embedded software developers, the hardware independence provided via MiL and SiL environments offers a major advantage. Long before target hardware or prototypes are available, functions, individual software components, as even the complete software can be tested and validated when integrated on a virtual control unit. Using the full potential of computing power of a PC, debugging and error reproduction in MiL and SiL can be performed much faster than on ECUs. Time consuming procedures such as flash programming can be avoided completely. The left part in Figures 1 and Figure 3 represents the test object, which can be a controller model, an ECU function, or the control software with an abstraction of the final ECU (virtual ECU) or a real ECU, depending on the XiL level. This describes a common functional abstraction for XiL levels and thus enables a large set of test cases to be execution candidates for different test levels [23].

An important problem in using multiple simulation tools in an integrated simulation is that they tend to use many different models of computation [24, 25]. For example, discrete-event, discrete-time, continuous-time and synchronous dataflow are among the many MoCs used. Each MoC has a specific mechanism for time progression and event handling. The integration platform must be able to handle tools that use different MoCs in a highly flexible manner. The integrated system must respect time synchronization with other simulation tools and preserve the causality of events. In addition to system integration, the platform must also enable integration of models by means of capturing the communication that occurs between them [25]. The XiL approach of ETAS addresses the aforementioned challenges by providing a solution to the following problems: (i) The approach decides on the appropriate FMI variant adapted for our multi-domain simulations. (ii) The approach provides the orchestration of the different FMUs in the simulation framework. (iii) It supports testing throughout the different XiL levels and facilitates the reuse of test cases. These three aspects are explained in the rest of this section.

4.1 FMI for a Hybrid Simulation

There exist two variants of the standard: FMI for model exchange (FMI-ME) and FMI for co-simulation (FMI-CS). Simulation in FMI-CS is meant to cover classical scenarios of co-simulation, where simulation models include their own solvers and exchange of data between simulation models is restricted to communication points. In this case, simulation models are solved independently between communication points. In FMI-ME, solvers of the importing simulation tool are used to solve the FMUs. In the meanwhile, the simulator has more options to handle synchronization between simulation models. This is due to the fact that in FMI-ME, time is described with super-dense time and there are two simulation modes for each simulation step that handle continuous and discrete model parts separately [26]. For a systematic workflow integrating the different components needed (in control development for our case) to seamlessly evolve through the different XiL levels. FMI-ME better fits our requirements, where continuous-time, discrete-time and event-triggered models are integrated to a hybrid simulation.

4.2 FMUs orchestration

Although the FMI standard enables the exchange of dynamic simulation models between different simulation tools, the specification itself does not define how simulation models get synchronized during micro- and macro-simulation steps. It only defines simulation procedure for each model within one simulation step. Master algorithms, which orchestrate the different heterogeneous simulation models, are mentioned but not further developed by FMI, and thus each simulation environment based on the FMI standard should

provide a master algorithm which orchestrates the entire simulation, synchronizes the simulation step sizes, or iterates discrete events. It is up to the tool vendors to offer solutions specifically suitable for their domains.

Continuous dynamics of powertrain systems can be modeled as a differential equation, a lookup table, or a multivariable polynomial. Discrete actions can be formulated as discrete events. The latter (recognition and handling of occurrences of asynchronous events) is an important case that needs to be supported when testing the ECU software. Using dynamic step size solvers in continuous-time models, it is possible to accurately identify the point in time when an event gets generated from the plant, by checking the event indicators. In this case, simulation step size might need to be adjusted to catch the event occurrence.

In MiL, both the controller and plant are modeled in continuous time domain, whereas in SiL the software implementation of the controller relies on the discrete-event domain. However, for the simulator itself, it shall be transparent whether the SiL model already contains ECU software on integration level where a complete operating system (OS) is configured with several OS tasks, or just the functionality of one function [26]. We follow the approach of [24] by using a model of computation (MoC) which describes the flow of data as well as the flow of control of simulation models for both continuous-time and for discrete-event domains .

4.3 Testing throughout XiL

The complexity of mechatronic systems leads to particularly long development and testing cycles. The long time span between the availability of the first software components and the first system prototype (e.g., an automobile) highlights the importance of testing earlier on in the development process. One of the main benefits of the XiL testing is to shift a big amount of cost-intensive test to an earlier stage, this then results in higher test coverage and enables incremental testing and intermediate validation, verification and correction. Testing under XiL platforms can be done as early as when the models are available (MiL testing). [23] demonstrates that there are significant similarities between the functional test cases across XiL levels. Consequently, a large set of test cases are execution candidates for different test levels, provided that they are specified at a reasonable functional abstraction. The need to create a comprehensive virtual test environment is well understood by ETAS. The XiL Approach targets the 3 viewpoints of test reuse identified by [27], namely the *vertical*, the *horizontal* and the *historical* case. The vertical case is concerned with the reuse between testing levels or types (e.g., component and integration testing, functional and performance testing). The horizontal case is concerned with the reuse between products in the same domain or family (e.g., standardized test suites, tests for product families, or tests for product lines). Finally the historical case is concerned with the reuse between product generations (e.g., regression testing).

For the design of reusable test cases throughout XiL, we follow a simulation based technique based on *black box* testing method (also referred to as *functional testing* or *behavioural testing*). Horizontal and historical reuse are achievable by standardization (FMI-compliant in our case). In principle, the controller in XiL exhibits the same logical functionality through the different levels. For functional testing through XiL, we also have the same interface but different technological constraints and access methods. The reuse of test cases is realized by means of the mapping components that bridge the technological gap as well as the abstraction gap between XiL levels. To achieve the vertical reuse, we opt for abstract test cases. These are considered as a formal basis for deriving executable test cases as they describe only the test scenarios. For the execution of the abstract test case, we use the mapping between the stimuli and the actual input (or signal), which is recognized by the system. The output of the test case should also be mapped onto the actual output of the system.

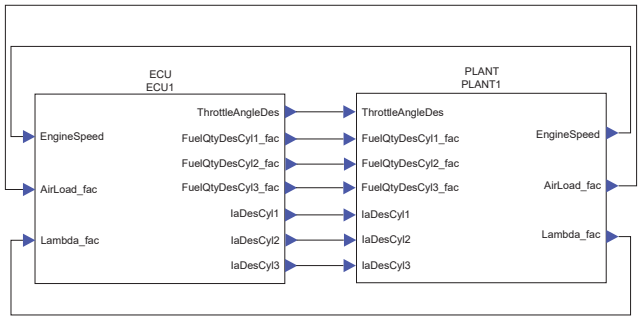

Figure 4: Composition of the internal combustion engine (ICE) model and the engine control unit ECU model.

5. RESULTS

To evaluate the performance of the XiL approach and framework described in this paper, we use a concrete industrial application as a case study. The case study is a 3-cylinder engine management System (Figure 4), consisting of three subsystems, namely a controller (ECU), the plant (engine model) and a simplified driver pedal model. The driver pedal subsystem has a control switch as input that turns to 1 when driver presses the acceleration pedal or remains at 0 when not pressed. In practical case, as will be explained later in the varying driver demand case, a driver may not maintain constant acceleration pedal position. This characteristic is modeled internally using a simple pulse generator for a specific case of 20% pedal position, i.e., the on-time of the pulse (T_{on}) corresponds to the time driver applies 20% acceleration and off-time of the pulse (T_{off}) is the time driver refrains from acceleration. The controller is responsible for fresh air load prediction, torque calculation and ignition, injection in the cylinder chamber. The internal computational unit (angle) is based on engine speed (EngineSpeed in Figure 4), which is a mandatory input for functioning of the subsystem. The other inputs to the subsystem are current air load factor (AirLoad_fac in Figure 4) in the cylinder and the current air/fuel ratio (Lambda_fac in Figure 4). The controller after air and fuel prediction actuates desired control by ignition angles (IaDesCyl1 to IaDesCyl3)

and injection fuel quantity (FuelQtyDesCyl1_fac to FuelQtyDesCyl3_fac) values. The engine subsystem responds to the inputs from the controller by developing combustion at the recommended injection quantity and ignition angles. The air/fuel mixture results in torque building inside the cylinder. The state of the engine can be understood by observing the current engine speed, air/fuel ratio and the air load factor in the cylinder. These outputs from the engine are fed back to the controller in order to complete the feedback control-loop system.

To illustrate the stable integration of heterogenous artifacts and the seamless transition between XiL levels (here we focus only on MiL and SiL) of our approach, we test our system under two different scenarios, namely the *idle speed control* case and the *varying driver demand* case. For each of these two cases, we record and compare the simulation results of the following setups:

- MiL using Simulink models of the plant and the controller
- MiL using generated FMUs of the plant (from MATLAB/Simulink) and the controller (from ASCET)
- SiL using generated FMUs of the plant (from MATLAB/Simulink) and the controller (from ASCET) with an abstraction of the ECU hardware (from ISOLAR-EVE [30])

Idle speed control: The idle speed control is a very important feedback control strategy that defines the combustion stability of an engine. The combustion stability in our case is described by torque, emission (air/fuel ratio) and mass air flow rate. The phenomenon of engine stalling is prevented by careful control of undershoot/overshoot of the engine speeds within the desired limits.

The simulation results are shown in Figure 5. Here we start the simulation with gas pedal activation input at time 0. The engine speed settling near to 800 rev/min can be observed the figure, illustrating the idle control behavior. During the idle control, the air/fuel ratio must be maintained at a constant level for a stable combustion. In the beginning of simulation (Figure 5), the idle speed control loop inside the controller is inactive. Upon receiving the engine speed on the first iteration, the controller recognizes that there is no driver pedal request and turns on the idle speed control. This control loop is performed in micro seconds. Once idle control loop is active, the engine is carefully brought to its idle set point value of 800 rev/min. During this time period, the destabilization of engine can happen if the fuel quantity and ignition angle per cylinder are delayed at the engine subsystem (it can result in misfire leading to stalling). The results show a synchronized timed event exchange between combustion engine and the engine controller for the 3 simulation setups.

From Figure 5 we can observe that there is a good match between the result of the MiL simulation in Simulink (in green), the FMI-based MiL simulation (in blue) and the SiL simulation (in red). This can be seen from the engine speed trajectory (the middle plot of Figure 5) and the estimated air flow trajectory (the lower plot of Figure 5). The slight difference between the three setups is due to the one step

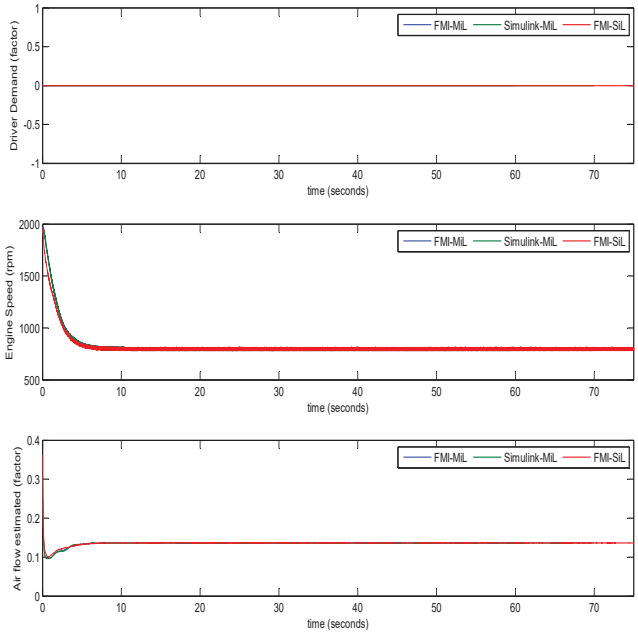

Figure 5: Idle speed control: comparison of simulation result in Simulink, FMI-based MiL and FMI-based SiL.

delay generated by the co-simulation [31].

Varying driver demand: The varying driver demand is a situation encountered by every driver during the course of a journey. The driver controls acceleration of vehicle by the gas pedal, which in turn modifies engine dynamics. The combustion stability continuously varies based on the changing driver demand and thereby introducing asynchronous event exchange. This relation can be seen on the variation of engine speed. Injection and ignition systems are highly sensitive to this asynchronous event exchange time, which may cause misfire and deteriorated engine performance.

For this case, we start the simulation with gas pedal activation input at 1 (gas pedal activation switch is pressed). This would mean that driver presses the gas pedal at 20% and removes creating a pulse. The engine speed increases rapidly to 5000rev/min on demand and comes back to idle when the foot is removed. During this test, the air/fuel ratio tries to remain close to a constant value for a stable combustion. The results in Figure 6 show that a synchronized timed event and asynchronous event exchange (air/fuel ratio, mass air flow and engine speed) between combustion engine and the engine controller is achieved. During the on-time of driver pedal demand, the engine speed slowly builds up to reach around 4750 rev/min and starts decrementing upon removal of accelerator pedal. The behavior is repeated for every cycle $((T_{on}) + (T_{off}))$. The engine cannot respond immediately to a driver demand and takes some time to reach its peak limit for the particular demand. This behavior can be observed in the Figure 6. Therefore during this test, frequent change (trigger of asynchronous events) in injection quantities and ignition angles ensure timely change (synchronized communication) of the engine speed in the en-

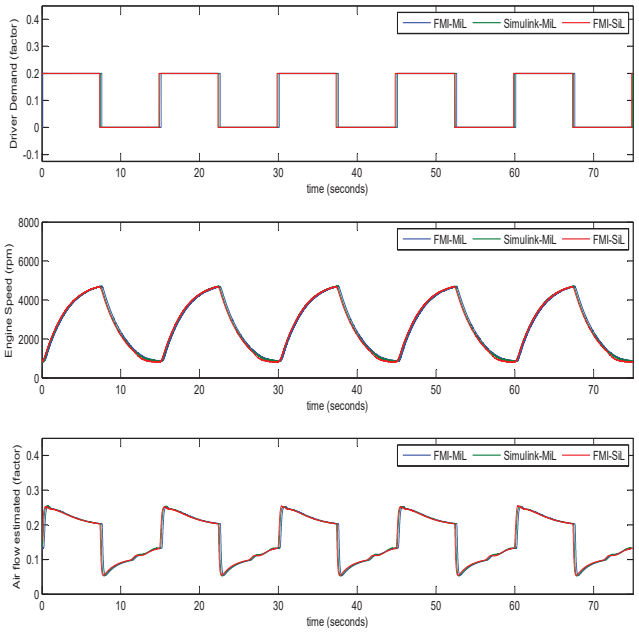

Figure 6: Varying driver demand: comparaison of simulation result in Simulink, FMI-based MiL and FMI-based SiL.

gine subsystem. Hence the test case shows a synchronized timed event and asynchronous event exchange between the combustion engine and the engine controller sub-system for the 3 simulation setups.

Here, similar to the idle speed control case, the result of the MiL simulation in Simulink, the FMI-based MiL simulation and the SiL simulation show a good match in both engine speed and air flow estimated.

The experimental results in both case show firstly that the FMI-based MiL simulation results of our approach is close to the native simulation results in Simulink. This means that our approach can deal with FMUs generated from different tools and achieve accurate simulation results. Secondly, we can also see that the simulation results between the FMI-based MiL setup and the SiL setup are similar, which shows that our approach can support seamless transition and test case reuse between different XiL levels.

6. CONCLUDING REMARKS

Testing and validation account for more than half of embedded system development costs. At the same time, process efficiency and product quality depend on test, validation, and calibration maturity. The sooner errors are identified in the development process, the higher the savings regarding bug fixing and refactoring-related effort and cost. This paper introduces ETAS XiL approach, which offers comprehensive virtualization solutions enabling testing, simulation and prototyping of embedded controls and controlled systems. By making a maximal use of open standards (FMI, ASAM XIL [29]) to achieve maximal integrability and reusability of artifacts throughout XiL levels. There are several possible future works in this direction. Although the use of standardized interfaces can assure the historical viewpoint of test reuse, it is only a valid statement as long as the system unter test has the same interface (inputs to be set and outputs to be assessed). Moreover, standards for parameterization and system structures [32] are still to be defined and adopted within our approach. Virtual and classical calibration shall be supported in exactly same seamless manner as testing. Parameterization variation and optimization at system level, as well as consolidations of already achieved results towards seamlessness, are just some of the aspects we mention as an outlook of the present work.

7. REFERENCES

[1] MATLAB/Simulink. MathWorks. http://de.mathworks.com/products/simulink/.

[2] ETAS ASCET. ETAS GmbH. http://www.etas.com/en/products/ascet_software_products.php?langS=true&/.

[3] Functional mockup interface for model-exchange and co-simulation. http://www.fmi-standard.org/.

[4] C. Dad, S. Vialle, M. Caujolle, J.P. Tavella, and M. Lanotto. Scaling of distributed multi-simulations on multi-core clusters. In *25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 142-147, June 2016.

[5] D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of FMUs for co-simulation. In *ACM International Conference on Embedded Software*, Sept. 2013.

[6] K. Nishimiya, T. Saito, and S. Shimada. Evaluation of functional mock-up interface for vehicle power network modeling. In *ACM/IEEE Design Automation Conference*, pages 1-6, June 2015.

[7] W. Chen, M. Huhn, and P. Fritzson. A generic FMU interface for Modelica. In *International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 19-24, Nov. 2011.

[8] Modelica. Modelica Association. https://www.modelica.org/.

[9] C. Bertsch. J. Neudorfer, E. Ahle, S. S. Arumugham, K. Ramachandran, and A. Thuy. FMI for physical models on automotive embedded targets. In *11th International Modelica Conference*, pages 21-23, Sept. 2015.

[10] M. Krammer, H. Martin, Z. Radmilovic, S. Erker, and M. Karner. Standard compliant co-simulation models for verification of automotive embedded systems. In *2015 Forum on Specification and Design Languages (FDL)*, pages 1-8, Sept. 2015.

[11] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automtoive software. In *Design, Automation, and Test in Europe (DATE)*, pages 1227-1232, Mar. 2012.

[12] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty. Multi-objective co-optimization of FlexRay-based distributed control systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1-12, April 2016.

[13] ITEA2 MODELISAR. https://itea3.org/project/modelisar.html/.

[14] S. Tripakis. Bridging the semantic gap between heterogeneous modeling formalisms and FMI. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*

(SAMOS XV), pages 60-69, July 2015.

[15] L. Exel, G. Frey, G. Wolf and M. Oppelt. Re-use of existing simulation models for DCS engineering via the functional mock-up interface. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1-4, Sept. 2014.

[16] M. U. Awais, P. Palensky, A. Elsheikh, E. Widl, and S. Matthias. The high level architecture RTI as a master to the functional mock-up interface components. In *2013 International Conference on Computing, Networking and Communications*, pages 315-320, Jan. 2013.

[17] E. Bringmann, and A. Kraemer. Model-based testing of automotive systems. In *International Conference on Software Testing, Verification and Validation*, pages 485-493, April 2008.

[18] A. Vidanapathirana, S. D. Dewasurendra, and S. G. Abeyarantne. Model in the loop testing of Complex Reactive Systems. In *International Conference on Industrial and Information Systems*, pages 30-35, Dec. 2013

[19] P. Caliebe, C. Lauer, and R. German. Flexible integration testing of automotive ECUs by combining AUTOSAR and XCP. In *International Conference on Computer Applications and Industrial Electronics*, pages 67-72, Dec. 2011.

[20] M. Shedeed, G. Bahig, M. W. Elkharashi, M. Chen. Functional design and verification of automotive embedded software: an integrated system verification flow. In *Electronics, Communications and Photonics Conference*, pages 1-5, April 2013.

[21] K. Hassani, W. Lee. A software-in-the-Loop simulation of an intelligent microSatellite within a virtual environment. In *International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications*, pages 31-36, July 2013.

[22] T. De Schutter. *Better Software. Faster!: Best Practices in Virtual Prototyping*. Synopsys, Inc., 2014.

[23] A. M. Perezand, and S Kaiser. Multi-level testing for embedded systems. *Model-Based Testing for Embedded Systems*. CRC Press, 2011.

[24] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming hetereogenity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127-144, Jan. 2003.

[25] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, and C. Sureshkumar. Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems. In *Proceeding of the 10th Modelica Conference*, pages 235-245, Mar. 2014.

[26] C. Stoermer, C. Mitrohin, A. Thuy, and G. Tibba. ETAS LabCar-XiL: bridging the gap between development phases by harmonizing concepts and tools. In *15. Internationales Stuttgarter Symposium: Automobil- und Motorentechnik*, pages 449-469, Mar. 2015.

[27] M. Karinsalo and P. Abrahamsson. Software reuse and the test development process: a combined approach. In *Proceedings of the 8th International Conference (ICSR)*, pages 59-68, July 2004.

[28] ETAS RT2. ETAS GmbH. http://www.etas.com/en/products/rt2.php?langS=true&.

[29] Generic simulator interface ASAM XIL. Association for Standardisation of Automation and Measuring Systems. https://wiki.asam.net/display/STANDARDS/ASAM+XIL.

[30] ISOLAR EVE. ETAS GmbH. www.etas.com/de/products/isolar_eve.php.

[31] M. Hu, Y. Huang, C. Zhao, X. Di, B. Liu, and H. Li. Model-based development and automatic code generation of powertrain control system. In *Model-based development and Automatic Code Generation of Powertrain Control System*, pages 1-4, Aug. 2014.

[32] Modelica association project system structures and parameterization. https://modelica.org/projects/