

Chapter 2

Standard Compliant Co-simulation Models for Verification of Automotive Embedded Systems

Martin Krammer, Helmut Martin, Zoran Radmilovic, Simon Erker,
and Michael Karner

2.1 Introduction

Cooperative simulation, or co-simulation, has become a common method to support the development of automotive systems. The integration of different modelling languages, tools and solvers into one common co-simulation enables new possibilities for design and verification of complex systems. Efforts to standardize the exchange of simulation models and enable integration in co-simulation scenarios were undertaken by the ITEA2 MODELISAR project. One of its main goals was the development of the functional mock-up interface¹ (FMI) [5, 14]. The FMI is an open standard which defines an interface supporting model exchange between simulation tools and interconnection of simulation tools and environments. The second version of the FMI standard was released in 2014 [6, 15].

SystemC² [19] is a C++ based library for modelling and simulation purposes. It is intended for the development of complex electric and electronic systems. SystemC targets high abstraction level modelling for fast simulation [2]. It provides sets of macros and functions, and supports paradigms like synchronization, parallelisms, as well as inter-process-communications. Its simulation engine is included in the library, and is built into an executable during model compilation. While

¹<http://www.fmi-standard.org>.

²<http://www.accellera.org>.

M. Krammer (✉) • H. Martin • Z. Radmilovic • S. Erker • M. Karner
Virtual Vehicle, Inffeldgasse 21a, 8010 Graz, Austria
e-mail: martin.krammer@v2c2.at; helmut.martin@v2c2.at; zoran.radmilovic@v2c2.at;
simon.erker@v2c2.at; michael.karner@v2c2.at

SystemC is capable of modelling and simulating digital systems, its SystemC-AMS³ extension expands these concepts to the analog and mixed signal domain. Both, SystemC and SystemC-AMS libraries, provide a certain degree of protection of intellectual property, when optimized and compiled models are exchanged.

In this work, we present a tool-independent method on how to integrate electric and electronic system models together with their corresponding simulation engines into single functional mock-up units (FMU) implementing the FMI. Aforementioned models are built using SystemC and SystemC-AMS. By doing so, SystemC becomes available to a broad range of applications on system level in a standardized manner. The resulting FMUs are highly transportable and may easily be integrated within larger and more complex co-simulation scenarios for fast and convenient information exchange and system verification.

This paper is structured as follows. Section 2.2 recapitulates related work. Section 2.3 characterizes relevant frame conditions and requirements. Section 2.4 introduces necessary steps on how to process models for FMU integration. Section 2.5 highlights the application of the proposed method with an automotive battery system use case. Section 2.6 summarizes the results and concludes this paper.

2.2 Related Work

Since the release of the FMI standard version 1.0 in 2010 and version 2.0 in 2014, efforts have been spent in order to implement and test the functional mock-up interface, in order to build new workflows for simulation and verification of systems under development. This section presents related work in the area of FMI, FMU generation, as well as parsing and usage in simulation scenarios.

Corbier et al. [10] introduces the FMI and argues about the necessity to share models for model/software/hardware-in-the-loop testing activities. As part of that a methodology for gradual integration and progressive validation is proposed. It also emphasises the need for conversion of existing models into the FMI standard.

Chen et al. [9] discusses technical issues and implementation of a generic interface to support the import of functional mock-up units into a simulator. For this import, the FMI calling sequence of interface functions from the standard are used.

Noll and Blochwitz [27] describes the implementation of FMI in SimulationX. It presents code generation out of a simulation model for FMUs for model exchange and co-simulation. A code export step generates the necessary C-code for model exchange. For co-simulation, a solver is included in the resulting dynamic link library (DLL). The tool coupling using SimulationX is accomplished by using a wrapper.

³<http://www.systemc-ams.org>.

The need for co-simulation in connection with the design of cyber-physical systems is highlighted in [32]. It follows the idea, that coded solvers in FMUs have some limitations regarding analysis or optimization. Therefore the authors strive for explicitly modelled ordinary differential equation solvers and claim a significant performance gain.

In [7] a verification environment using Simulink and SystemC is introduced. It relies on s-functions to create a wrapper in order to combine SystemC modules with Simulink.

In [30] the generation of FMUs from software specifications for cyber-physical systems is outlined. This approach fulfills the need for software simulation models. A UML based software specification is automatically translated into a FMU, maintaining its original intended semantics. This step is done using C-code, which is included within the FMU.

In [25] a high level approach for integration and management of simulation models for cyber-physical systems is shown.

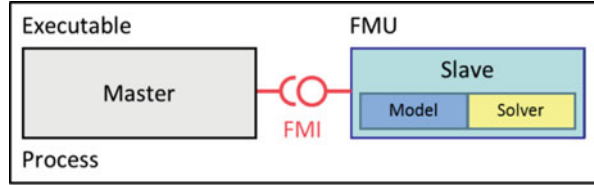
Elsheikh et al. [13] presents an integration strategy for rapid prototyping for Modelica models into the FMI standard, and highlights a high level approach for integration of cyber-physical systems.

SystemC and SystemC-AMS are used in a variety of simulation platforms, where different wrappers or adapters provide data exchange services. Examples thereof are given in [1, 20]. Regarding the use of SystemC or SystemC-AMS in the context of the FMI standard, no relevant publications are available to date, describing a unified process for integration. Thus, the outlined approach of integrating SystemC/SystemC-AMS models together with the functional mock-up interface into a FMU is considered as a novel contribution to the field of applied co-simulation.

2.3 Requirements on Modelling and Co-simulation Data Exchange

In this section, general framework requirements and specifics are captured. This affects the SystemC and SystemC-AMS languages and libraries, the FMI standard, as well as co-simulation specific aspects. Usually, the compilation of SystemC or SystemC-AMS models leads to a platform specific executable, containing all models as well as necessary schedulers and solvers. This property seems suitable for application of SystemC's modelling and simulation concepts in context of the FMI. In order to comply to the FMI standard for co-simulation, a dynamic link library implementing the FMI needs to be compiled and assembled instead of an executable. The targeted FMI application scenario can be found in [14, 15] and is shown in Fig. 2.1 for one single FMU. One co-simulation master is expected to coordinate the co-simulation by utilizing the FMI for communication to the FMU. The FMU

Fig. 2.1 Co-simulation with generated code on a single computer [15]



includes the entire model and the corresponding scheduler or solver. SystemC and SystemC-AMS based approaches differ from co-simulation with tool coupling, as no separate tool is required for simulation execution.

A state machine for the calling sequence of the FMI standard co-simulation interface C-functions is available [14, p. 31]. For a basic repetition of simulation steps, the following principles can be distilled. First, a FMU is subject to instantiation and initialization. Then, input values, parameters and states are set using corresponding `fmiSet[...]()` group of functions. This is followed by a simulation execution phase called using `fmiDoStep()` function. If that step completes, the `fmiGet[...]()` group of functions is used to retrieve the results for external communication. In scope of this work the previously mentioned functions need to be implemented in SystemC and made available to the FMI based on a C-code interface. Since SystemC supports the concept of time, all C-interface function calls must be synchronized during co-simulation.

One further main criteria refers to the SystemC and SystemC-AMS libraries. Both libraries are available license free. SystemC is standardized under IEEE 1666 [19], whereas SystemC-AMS is documented in the Language Reference Manual version 2.0. Thus, it makes sense to strive for a solution which builds on these standard documents and does not cause any changes to the corresponding implementation libraries.

Under normal circumstances, SystemC and SystemC-AMS module simulations are performed in a single run, using the time domain simulation analysis mode. This means that the method `sc_start()` runs the initialization phase and subsequently the scheduler through to completion [11]. However, `sc_start()` may be called repeatedly with a time argument, where each simulation run starts where the previous run left off. For co-simulation, where data exchange and synchronization happens on discrete points in time, this function is vital to control simulation within the FMU. Memory management is rarely an issue in standard SystemC/SystemC-AMS models, due to their single elaboration phase and comparably short run times. In encapsulated FMUs, memory management can be crucial as the FMI standard suggests that FMUs have to free any allocated resources by themselves. The standard therefore defines instantiation and termination functions, therefore proper construction and destruction of all SystemC/SystemC-AMS models contained in FMUs is desirable to avoid any memory leaks.

The SystemC simulation kernel supports the concept of a delta cycle. One delta cycle consists of an evaluation phase followed by an update phase. This separation ensures deterministic behavior [11], as opposed to e.g. the use of events. Events

trigger process executions, but their execution order within one single evaluation phase is non-deterministic. The concept of a delta cycle is even more important when moving from simulation level to co-simulation level, where external signals are connected to the model. New values written to e.g. signals become visible after the following delta cycle. Execution of one delta cycle does not consume simulated time.

SystemC and SystemC-AMS feature four different models of computation (MoC). According to [34], a MoC is defined by three properties. First, the model of time employed. Second, the supported methods of communication between concurrent processes. And third, the rules for process activation. SystemC features a kernel including a non-preemptive scheduler [19], which operates discrete event (DE) based for modelling concurrency. This MoC is used for modelling and simulation of digital software and hardware systems. SystemC-AMS features three different MoC, which may be used depending on the actual application. Namely these are timed dataflow (TDF), linear signal flow (LSF) and electrical linear networks (ELN). TDF operates on samples which are processed at a given rate, with a specified delay, at a given time step interval. TDF and DE MoC are synchronized using specified converter ports. LSF is primarily used for signal processing or control applications and features a broad range of predefined elements within the SystemC-AMS library. Converter modules for the conversion to and from the TDF and DE MoC exist. ELN permits the description of arbitrary linear networks and features an element library as well. Converter modules for the conversion to and from the TDF and DE MoC exist. Our goal is to support all four MoC in context of simulation through the FMI.

2.4 Model Integration Method

In order to integrate and execute SystemC and SystemC-AMS simulation models in context of an FMU, we propose a structured method. The necessary steps are illustrated in Fig. 2.2, indicated by the dashed box. They are described as follows.

- (A) Modelling and simulation of a single component model.
- (B) Model interface identification for coupling to the co-simulation environment.
- (C) Wrapper class specification for controlling the model interface.
- (D) C-interface specification for FMI integration.
- (E) FMI integration using a predefined software developer kit.
- (F) FMU compilation and assembly together with (architectural) model description.
- (G) Integration of FMU to co-simulation master for simulation based system level verification.

Subsequently, each step is explained in detail.

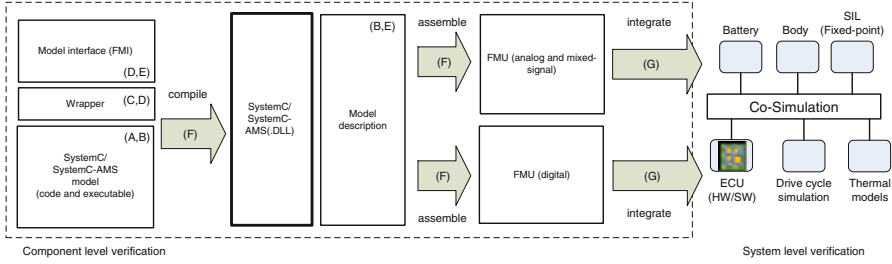


Fig. 2.2 The proposed process for the integration of executable simulation models into FMUs for co-simulation

2.4.1 Modelling and Simulation

To embed a SystemC or SystemC-AMS simulation model in a FMU, the model has to be set up and tested against its specifications first. Standalone executables may be executed, traced, and debugged using additional tools like an integrated development environment. However, a proprietary co-simulation master usually does not offer such sophisticated debug possibilities for compiled and assembled FMUs. To instantiate and test the model under development, a test bed is typically used. It may consist of stimuli generators, reference models or watchdogs [16]. The *SystemC Verification Library* (SCV) [35] is also available for this purpose.

2.4.2 Model Interface Identification

In a second step, the interface of the simulation model, which is later exposed through the FMI, shall be determined. This includes the definition of input and output quantities, or states, as well as internal timing (accuracy and precision required by the simulation model) and external timing (simulation step size for data exchange considerations). If a system level design is available, the model interface identification can be accomplished using these specifications.

2.4.3 Wrapper Class Specification

Typically, `sc_main()` is used to indicate the top-level module and to subsequently construct the entire module hierarchy through instantiation. The latter happens during the so called elaboration phase, right before the execution of `sc_start()`. For co-simulations, the breakdown of time into time steps is achieved by calling `sc_start()` multiple times. Thus it is necessary to keep the entire module hierarchy and its states persistently in memory, as seamless data exchange between

simulation steps must be ensured. This can be achieved by using a C++ wrapper class defining a constructor and destructor managing the top-level SystemC simulation model in memory, until the entire co-simulation is finished. Listing 2.1 shows an example constructor for a SystemC wrapper class. Additionally, the wrapper class has `sc_signal` primitive channels attached. These are used for realization of the interface identified in step (B).

```
BatteryControllerWrapper::BatteryControllerWrapper() {
    controller = new BatteryController("batteryController");

    signalVoltageIn = new sc_signal<double>;
    signalSocOut = new sc_signal<double>;
    controller->in_voltage(*signalVoltageIn);
    controller->out_soc(*signalSocOut);
}
```

Listing 2.1 Wrapper class constructor.

2.4.4 C-Interface Specification

The main idea here is to have a set of functions, which can be used to initialize, control, and finally shut down the C++-based SystemC/SystemC-AMS simulation model within a C-based FMU. To bridge the gap between the C-language FMI and the C++-language based simulation models, special linker instructions are required when compiling the file for the FMU. The compiler keyword used for this purpose is `extern "C"`. This C++ standard feature is a linkage-specification every compiler is required to fulfill. It exposes the enclosed C++ functions to the FMI.

For SystemC/SystemC-AMS, this means that signals (e.g. `sc_signal`) or ports (e.g. `sc_in/sc_out`) may be used for variable modifications when the scheduler or simulation engine is not running. However, the scheduler requires the execution of one delta cycle to adopt a value which is written to a signal, otherwise the previous value remains assigned. This is achieved by using the `SC_ZERO_TIME` macro. It updates the signal's value while it does not advance simulation time.

For SystemC-AMS, the different MoC are combined advantageously using converters, in order to get and set input and parameter values as desired. To couple e.g. an electric current, the `sca_eln::sca_tdf_isink` and `sca_eln::sca_tdf::sca_r` primitive modules from the ELN MoC library may be used to read or write electric current values using the TDF MoC, respectively. A code example for setting a value to a TDF MoC module can be seen in Listing 2.2. Again, for execution of one delta cycle, the `SC_ZERO_TIME` macro is used.

```
extern "C" void setBatteryCurrent(double current) {
  wrapper->batteryModule->cellGenerator->setCurrent(current);
  sc_start(SC_ZERO_TIME);
}
```

Listing 2.2 Assignment of a value to a TDF module method

For FMI integration, a minimum of six different kinds of functions are required. `startInterface()` is required as an entry point to the SystemC/SystemC-AMS model, when simulation is initialized. Called once per FMU instantiation, this function calls `sc_main()` for the first time. One delta cycle is increased by calling `sc_start()` with the `SC_ZERO_TIME` argument. This triggers the construction of the simulation model in memory via the previously introduced wrapper class from step (C). This ensures that the simulation model is completely hierarchically constructed in memory and ready for simulation, without any simulated time passing by yet. `shutdownInterface()` is used to destruct all impressed models and free the occupied memory. `setValueXXX()` and `getValueXXX()` functions are used for each coupled variable to pass values through the FMI directly to the SystemC/SystemC-AMS model. The `doSim()` function is used to trigger the simulation start. It basically calls `sc_start()` with a SystemC time format parameter. If `sc_start()` is called using a fixed time interval, this time interval represents the step size of the FMU. In order to dynamically pass a required time interval setting to the simulation model, the `setTimeStep()` function is used. This realizes an adaptive step size co-simulation. The time step value is stored within the wrapper class. The call to `sc_start()` is modified accordingly, as seen in Listing 2.3.

```
sc_core::sc_start(wrapper->timeStep, sc_core::SC_SEC);
```

Listing 2.3 Dynamic simulation step size assignment.

2.4.5 FMI Integration

For integration of the FMI, the FMU SDK (software development kit) is used as a basis [33]. It provides functions and macros which are included next to the SystemC/SystemC-AMS files. The main header file establishes a logical connection between the software code and the descriptive XML file that is required for each FMU. This `modelDescription.xml` file contains major information about the FMUs architecture. This also includes gathered information from step (B).

2.4.6 FMU Compilation and Assembly

Finally, the FMU parts are now compiled and assembled. According to [14, p. 38], a FMU is referred to as a zip file with a predefined structure. The .dll file is placed in the binaries folder for the corresponding platform. The modelDescription.xml file is placed in the top level (root) folder. The SystemC/SystemC-AMS source files are placed in the sources folder optionally. Corresponding model documentation or associated requirements (see [21] for details) may be placed in the documentation folder. Initialization values, like sets of parameters, may be placed in the resources folder. After creating a zipped file from these contents, the FMU is ready for distribution and instantiation.

2.4.7 FMU Integration for Co-simulation

The FMI for co-simulation standard defines a master software component which is responsible for data exchange between subsystems. In this work, the independent co-simulation framework (ICOS) from *Virtual Vehicle Research Center* [18] is used. Typically, it is applied to solve multidisciplinary challenges, primarily in the field of automotive engineering. Use cases include integrated safety simulation, electrical system simulation, battery simulation, thermal simulation, mechanical simulation, and vehicle dynamics simulation. It features an application design which separates the co-simulation framework and its coupling algorithms from the simulation tools that are part of the co-simulation environment. Hence the co-simulation framework is independent from the simulation tools it integrates [31]. The framework relies on the exchange of discrete time signal information, and provides several different algorithms e.g. for interpolation, extrapolation, and error correction methods [3] of these signals. The exchange of discrete time signals from one co-simulation component model to another is called coupling. The independent co-simulation framework implements the FMI standard and allows instantiation and co-simulation of FMUs. Alternatively, and to ensure FMI standard compliance, a *FMU checker* executable is provided with the standard. It instantiates an FMU and performs basic operations on it automatically.

2.5 Automotive Battery System Use Case

In order to demonstrate the functionality of the resulting FMUs described in Sect. 2.4, an automotive battery system use case was selected. The battery system introduced here is intended for a hybrid electric vehicle (HEV), where the battery powers an electric motor next to a combustion engine, as main components of the power train. Several strategies are known to charge and discharge an automotive

battery propelling a vehicle. Information about driver behavior, routing, road profile, etc. have a strong influence on the behavior of the battery system.

Such a battery system usually consists of two main components, namely the battery pack as energy storage facility and a corresponding battery controller. The targeted battery pack consists of four battery modules, where each of them integrates 12 cells with a nominal capacity of 24Ah each. The targeted controller monitors operational condition and health of each of the modules. The controller also communicates with the car's hybrid control unit (HCU) to ensure stable vehicle operation. For the battery module, we construct a SystemC-AMS based FMU utilizing the ELN and TDF MoC. For the battery controller, we implement a SystemC/SystemC-AMS based FMU utilizing the DE and LSF MoC. Both FMUs are integrated into one common co-simulation scenario.

2.5.1 Battery Module FMU

The success of HEVs strongly depends on the development of battery technology. For the development of battery based systems it is essential to know the characteristics and the behavior of the battery. Especially in the field of functional safety, the knowledge of these features is key to control potential hazards. There is a wide range of different existing modes available for simulating the performance of lithium-ion battery cells. Most common approaches are electro-chemical models [12, 26, 29] and equivalent circuit models (ECM) [4, 8, 17, 24].

Electrochemical models describe the internal dynamics of a cell leading to complex partial differential equations with a large number of unknown parameters. Since these models are computationally expensive and therefore time consuming, they are not suitable for system-level modeling. ECMs on the other hand are computationally more efficient and are therefore better suitable for system-level modeling. Such models are also commonly used in embedded battery management systems (BMS) to estimate the state of charge (SoC) and predict the performance of physical batteries. An ECM is composed of primitive electrical components like e.g. resistors, capacities and voltage sources to simulate a cell's terminal voltage response to a desired current flow. It is capable to accurately describe the static as well as the dynamic behavior of a cell under various operating conditions. In [36] an ECM model of a battery cell intended for portable devices, written in SystemC-AMS, is shown.

For this use case, we model an ECM according to [24, 36]. The corresponding circuit diagram is shown in Fig. 2.3. Such a two- RC block model (a) is a common choice for lithium-ion cells [17]. These two RC blocks characterize short-term and long-term dynamic voltage response of the cell, respectively, which arise from diffusion phenomena in the cell. I_{bat} in (b) represents an identified input current to the FMU according to Sect. 2.4. The controlled voltage source V_{OC} reproduces the open circuit voltage (OCV), which represents an output of the FMU according to Sect. 2.4. The serial resistor R_0 describes the internal resistance of the cell comprised

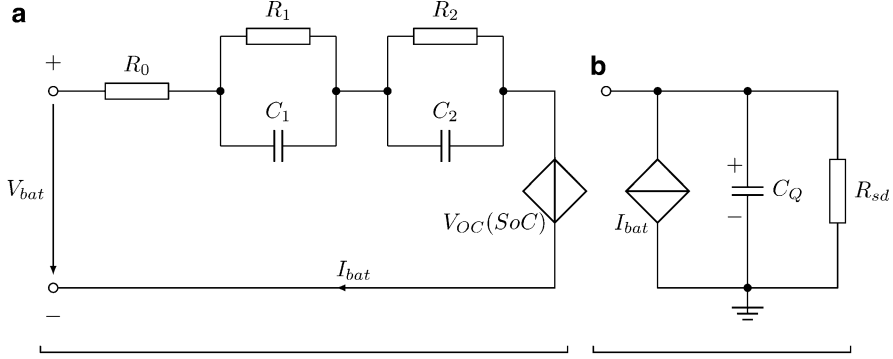


Fig. 2.3 Schematic diagram of the used cell equivalent circuit model: (a) voltage-current characteristics, (b) energy balance circuit

of ohmic and charge transfer resistances and is connected in series with the two RC branches. In general, all parameters of the model depend on several quantities like state of charge (SoC), temperature, cell age as well as current direction and rate.

Mathematically the electrical behavior of an ECM with two RC branches can be expressed by Eqs. 2.1 and 2.2.

$$V_{bat} = V_{OC} + V_1 + V_2 + R_0 I_{bat} \quad (2.1)$$

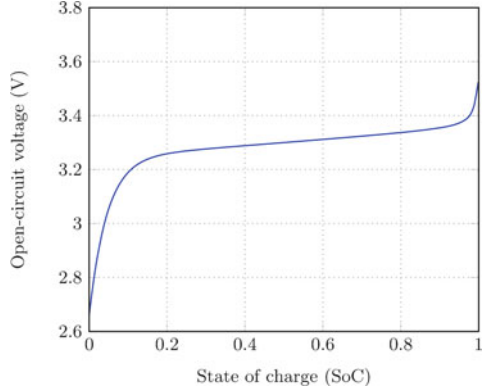
$$\begin{bmatrix} \dot{V}_1 \\ \dot{V}_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{R_1 C_1} & 0 \\ 0 & \frac{1}{R_2 C_2} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{C_1} \\ \frac{1}{C_2} \end{bmatrix} I_{bat} \quad (2.2)$$

with $V_{1,2}$ and $\dot{V}_{1,2}$ the voltages across the RC branches and their time derivatives, respectively.

To parametrize an ECM, measurements on physical batteries are often performed to create large multidimensional look-up tables for the various parameters to cover all their dependencies. The lithium-ion battery model is based on data of a LiFePO₄ (LFP) battery [24]. To limit complexity for this use case we propose a simplified model, where the OCV depends on the SoC, $V_{OC} = f(\text{SoC})$, and all other parameters are considered constant using averaged values from the data of [24]. The used $\text{SoC} - V_{OC}$ relationship is based on [23] and was adapted for usage out of its ordinary range to support simulation of e.g. overcharge scenarios. Figure 2.4 shows the $\text{SoC} - V_{OC}$ relationship. However, the model can straightforwardly be extended to a more advanced model by including additional elements representing terms for e.g. thermal behavior or aging effects. Thermal models may also be attached through co-simulation later on.

It is noteworthy that the ECM model can be adopted for other battery chemistries by simply adjusting the set of parameters used in the model. In context of a FMU, these could be placed within the FMU's resources folder in e.g. XML file format for model configuration during instantiation.

Fig. 2.4 Battery open circuit voltage V_{OC} as a function of SoC



From a black box view, the proposed battery module FMU shall support the following functionality.

- One module shall consist of 12 cells in series connection, each modelled as described in this section.
- Current demand is an input to the battery module.
- Voltage is an output of the battery module.
- The battery module's SoC shall be set from external prior to simulation (without recompilation of the FMU).

A battery module with these properties is modelled in SystemC-AMS as follows. The ECM of a battery cell shown in Fig. 2.3 representing Eqs. 2.1, 2.2 is implemented using primitives of the ELN MoC of SystemC-AMS. The TDF MoC is used to interface the ECM, e.g. to control the voltage source V_{OC} the class `sca_tdf_vsource` primitive is used. The cell is instantiated 12 times, and these instances are connected in series to model a single battery module. The battery module is subject to integration into a FMU according to Sect. 2.4.

2.5.2 Battery Controller FMU

The electrochemical processes inside a battery are considered complex and the lithium-ion cell operating window is narrowed down by voltage and temperature restrictions. A battery controller monitors cells and modules to keep the battery pack and the entire vehicle functional within a safe state. Monitoring measures for batteries typically capture the state of charge (SoC), cell voltages as well as the temperature. The determination of the SoC is very complex, since it cannot be measured directly. Many vehicle applications require an exact knowledge of the SoC of the battery. The most obvious include the calculation of the vehicle's remaining driving range. More sophisticated applications include communications to the HCU, e.g. recuperation functions, influence on driving modes, trip routing, or

comfort functions. Several solutions to the problem of accurately estimating the SoC have been proposed in literature [28]. The most common method for calculating the SoC is coulomb counting, which is based on measuring battery current. With the knowledge of an initial SoC_0 , the remaining capacity in a cell can be calculated by integrating the current that is entering (charging) or leaving (discharging) the cell over time:

$$SoC = SoC_0 + \frac{1}{C_Q} \int_{t_0}^t \eta I_{bat}(\tau) \cdot d\tau \quad (2.3)$$

Here C_Q is the rated capacity (the energy capacity of the battery under normal condition), I_{bat} is the battery current and η is a factor that accounts for loss reactions in the cells (we assume $\eta = 1$). Coulomb counting is straightforward to implement and able to determine the SoC under load, which makes it suitable for on-board applications. However, it requires the initial SoC of the battery. To get SoC_0 the cell voltage under no-load is measured and from this the SoC can be determined from the $SoC - V_{OC}$ relationship.

We propose a battery controller FMU, implementing four main functionalities:

- Sampling of the battery voltage.
- Sampling of the battery current.
- Calculation of the SoC based on $SoC - V_{OC}$ relationship and look-up table.
- Calculation of the SoC based on coulomb counting using current integration.

The battery controller is realized as a SystemC/SystemC-AMS module. It uses one thread for periodical voltage sampling and one for look-up table operations (based on the relationship depicted in Fig. 2.4). It utilizes the integrator primitive from the LSF MoC of SystemC-AMS for current flow calculation according to Eq. 2.3. In the end, the module is compiled and assembled as an FMU according to Sect. 2.4.

2.5.3 Co-simulation Integration

The resulting FMUs pass the *FMU checker* test and are integrated into the independent co-simulation framework as described in Sect. 2.4.7. First, its boundary condition server (BCS) is used to test the resulting single FMUs. Second, the FMUs are integrated into a co-simulation scenario. This includes a mild hybrid vehicle together with a drive cycle modelled in CarMaker, and the hybrid controls modelled in Matlab Simulink.

The integration is accomplished using a computer aided software engineering (CASE) tool, namely Enterprise Architect.⁴ It uses a co-simulation framework

⁴<http://www.sparxsystems.com>.

specific extension [22] based on UML and SysML languages to integrate the required simulation models, assign configuration information to them, and generate a co-simulation configuration file out of the underlying model repository. For model integration, the model interface information described in Sect. 2.4 (B) is used. The simulation models and their couplings are represented in a co-simulation architecture diagram (cad). This is shown in Fig. 2.5. The couplings between models are defined using relationships at their interfaces. In order to ensure the correct transformation from the SysML model to the co-simulation configuration file, a model analyzer checks for presence of modelling errors. The co-simulation itself is executed through the independent co-simulation framework, which consumes the generated configuration file.

2.5.4 Discussion of Results and Observations

The following Table 2.1 provides an overview of different simulation scenarios and their achieved performance within the independent co-simulation framework as described in Sect. 2.4.7. The simulated time is denoted as t_s , whereas t_e refers to the time needed for simulation execution on a standard laptop device.

Figure 2.6 shows the output of the controller's estimation of the SoC. In this scenario, the module is discharged with 20 A current pulses with a pulse period of $T = 1100\text{ s}$ and a duty factor of $\tau/T = 1/11$. State of charge estimation with a simple OCV-SoC lookup table does not make sense under load conditions, as during times $T - \tau$. Allowing a sufficiently large relaxation time after a discharge pulse we can compare the two methods and yield similar results for the SoC. Next, the controller and battery module FMUs are coupled to the vehicle model and its hybrid controls. Figure 2.7 shows an excerpt of the electric motor current demand and battery module state-of-charge as observed during a drive cycle.

From a qualitative point of view, the battery simulation results correspond to the results described in [23]. Quantitatively, the generated battery module FMU reproduces the simulation results shown in [36], validating against Li-Po cells from [9]. For this case no relevant increase of simulation time caused by the co-simulation framework was measured. The step size considered for co-simulation was 1 s.

The ascertained overall efforts for FMU integration are considered justifiable, once the co-simulation interface has been defined. However, the following issues should be observed when following the proposed process. By encapsulating simulation models into FMUs, an additional layer of time synchronization is introduced. FMU-internal SystemC/SystemC-AMS module step sizes may take very small values and account for precise calculations. In contrast to that, a very small external co-simulation step size causes increased coupling-related communications, produces vast amounts of data, and therefore slows down simulation performance. From this it follows that the internal and external step sizes used may not diverge by higher orders of magnitude.

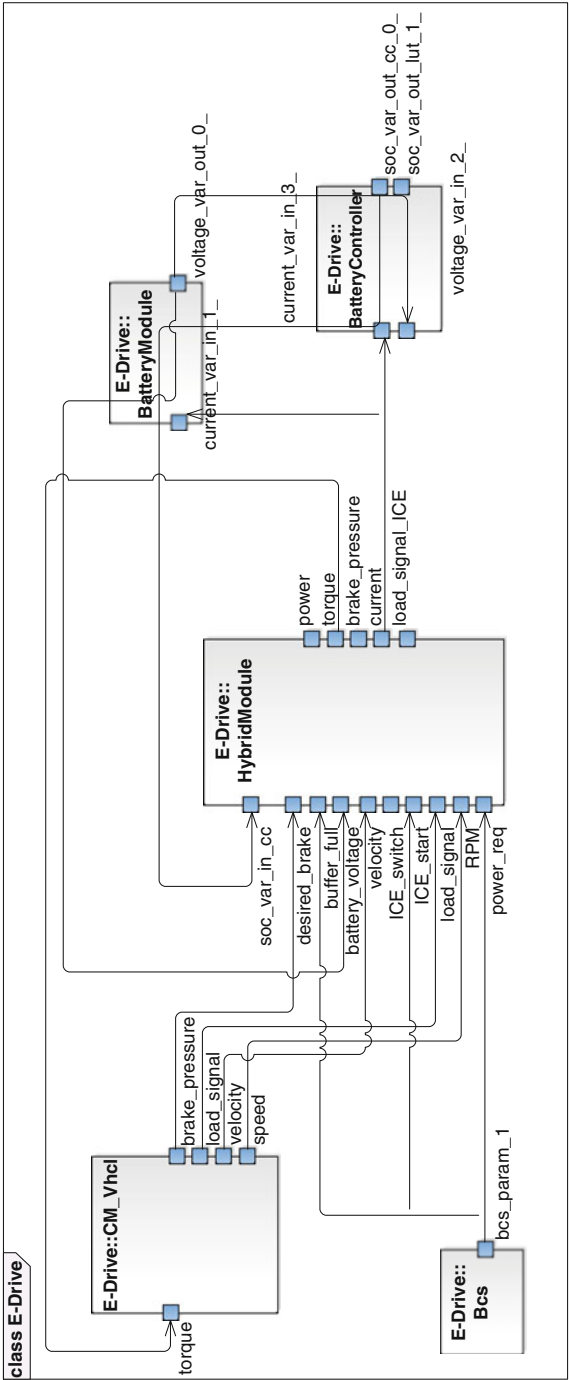


Fig. 2.5 Model integration

Table 2.1 Simulation time evaluation

Scenario	t_s [s]	t_e [s]
Battery controller w/ discharge pulses (BCS)	8000	9
Battery controller w/ drive cycle current (BCS)	200	<1
Battery module w/ discharge pulses (BCS)	8000	2
Battery module w/ drive cycle current (BCS)	200	<1
Battery module, controller w/ discharge pulses (BCS)	10, 000	15
Battery module, controller w/ discharge pulses (BCS)	20, 000	28
Mild hybrid vehicle drive cycle	200	17

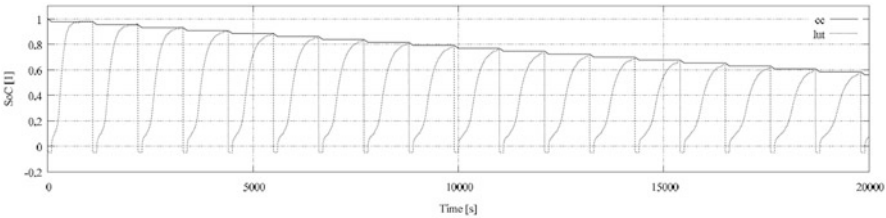


Fig. 2.6 Estimation of the state-of-charge using voltage based look-up table (lut) and coulomb counting with current integration (cc) approaches. For this simulation, a 20 A pulse discharge test was conducted

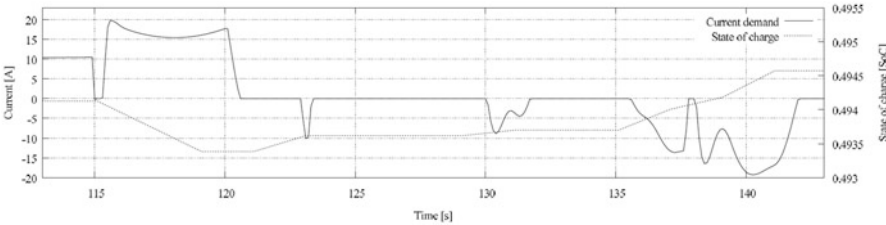


Fig. 2.7 The electric motor current demand (*solid line*) and battery module state-of-charge (*dashed line*) of a hybrid electric vehicle as observed during a drive cycle. The co-simulation scenario includes a vehicle power train model and a hybrid control unit model coupled to the two proposed FMUs

To ensure stable loading, execution, and unloading processes of FMUs at the FMI master, the use of pointers and dynamic memory allocation when constructing SystemC/SystemC-AMS modules is indispensable.

The FMI standard defines a resource folder inside a FMU, which may be used for e.g. different sets of ECM parameter settings. This is ideal for model exchange scenarios where models are kept separately from their associated parameters and configurations.

The creation of the required FMU XML file and synchronization to the model code causes additional efforts due to variable numbering and name assignments, especially if models are modified. Additional automation could help to improve these deficiencies, e.g. use of a model based software development approach.

2.6 Conclusion

In this paper a structured method for the integration of SystemC/SystemC-AMS simulation models to the FMI standard is introduced. The presented method does not require any changes to the standardized SystemC or SystemC-AMS libraries. The method eases the integration of existing system level simulation models into larger and more complex simulation scenarios, which are used for information exchange and verification on system level. A two-part battery system use case from the automotive domain is presented, which exploits these MoC for simulation. The resulting FMUs created with the described method are highly transportable and configurable. These properties make them suitable for verification and information exchange processes within the automotive domain.

Acknowledgements This research work has been funded by the European Commission within the EMC² project under the ARTEMIS Joint Undertaking under grant agreement no. 621429. The authors acknowledge the financial support of the COMET K2—Competence Centres for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Science, Research and Economy (BMWFW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

References

1. Armengaud, E., Karner, M., Steger, C., Weiß, R., Pistauer, M., Pfister, F.: A cross domain co-simulation platform for the efficient analysis of mechatronic systems. In: SAE World Conference (SAE Technical Paper 2010-01-0239), pp. 1–14 (2010). doi:[10.4271/2010-01-0239](https://doi.org/10.4271/2010-01-0239)
2. Barnasconi, M.: SystemC AMS extensions: solving the need for speed. In: Design Automation Conference (2010)
3. Benedikt, M., Watzenig, D., Zehetner, J., Hofer, A.: NEPCE - a nearly energy preserving coupling element for weak-coupled problems and co-simulation. In: IV International Conference on Computational Methods for Coupled Problems in Science and Engineering, Coupled Problems (2013)
4. Birkel, C., Howey, D.A.: Model identification and parameter estimation for *LiFePO4* batteries. In: Hybrid and Electric Vehicles Conference 2013 (HEVC 2013), p. 2.1–2.1. Institution of Engineering and Technology, Institution of Engineering and Technology, London (2013). doi:[10.1049/cp.2013.1889](https://doi.org/10.1049/cp.2013.1889)

5. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.V., Wolf, S.: The functional mockup interface for tool independent exchange of simulation models. In: 8th International Modelica Conference 2011, pp. 173–184 (2011). doi:[10.3384/ecp12076173](https://doi.org/10.3384/ecp12076173)
6. Blochwitz, T., Otter, M., Akesson, J.: Functional mockup interface 2.0: the standard for tool independent exchange of simulation models. In: NAFEMS World Congress (2013)
7. Boland, J., Thibeault, C., Zilic, Z.: Using MATLAB and simulink in a SystemC verification environment. In: Proceedings of Design and Verification Conference (2005)
8. Chen, M., Rincon-Mora, G.: Accurate electrical battery model capable of predicting runtime and I-V performance. *IEEE Trans. Energy Convers.* **21**(2), 504–511 (2006). doi:[10.1109/TEC.2006.874229](https://doi.org/10.1109/TEC.2006.874229)
9. Chen, W., Huhn, M., Fritzson, P.: A generic FMU interface for Modelica. In: 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, pp. 19–24 (2011)
10. Corbier, F., Loembe, S., Clark, B.: FMI technology for validation of embedded electronic systems. In: Embedded Real Time Software and Systems (2014)
11. Doulos: SystemC Golden Reference Guide. Doulos, Ringwood (2002)
12. Doyle, M., Fuller, T.F., Newman, J.: Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *J. Electrochem. Soc.* **140**(6), 1526–1533 (1993). doi:[10.1149/1.2221597](https://doi.org/10.1149/1.2221597)
13. Elsheikh, A., Awais, M.U., Widl, E., Palensky, P.: Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. In: 2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013, pp. 1–6 (2013). doi:[10.1109/MSCPES.2013.6623315](https://doi.org/10.1109/MSCPES.2013.6623315)
14. Functional Mock-up Interface for Co-Simulation, Version 1.0 (2010)
15. Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0 (2014)
16. Grotker, T.: System Design with SystemC. Kluwer Academic, Norwell (2002)
17. He, H., Xiong, R., Guo, H., Li, S.: Comparison study on the battery models used for the energy management of batteries in electric vehicles. *Energy Convers. Manag.* **64**, 113–121 (2012). {IREC} 2011, The International Renewable Energy Congress
18. ICOS Independent Co-Simulation - User Manual Version 3 (2013)
19. IEEE Standard 1666: SystemC Language Reference Manual (2011)
20. Krammer, M., Karner, M., Fuchs, A.: Semi-formal modeling of simulation-based V&V methods to enhance safety. In: Proceedings of the Embedded World 2014 Exhibition and Conference. WEKA Fachmedien GmbH, Nuremberg (2014)
21. Krammer, M., Karner, M., Fuchs, A.: System design for enhanced forward-engineering possibilities of safety critical embedded systems. In: 17th International Symposium on Design and Diagnostics of Electronic Circuits Systems, pp. 234–237 (2014). doi:[10.1109/DDECS.2014.6868797](https://doi.org/10.1109/DDECS.2014.6868797)
22. Krammer, M., Fritz, J., Karner, M.: Model-based configuration of automotive co-simulation scenarios. In: Proceedings of the 47th Annual Simulation Symposium. The Society for Modelling and Simulation International, San Diego (2015)
23. Lam, L.: A practical circuit-based model for state of health estimation of li-ion battery cells in electric vehicles. Ph.D. thesis, TU Delft, Delft University of Technology (2011)
24. Lam, L., Bauer, P., Kelder, E.: A practical circuit-based model for Li-ion battery cells in electric vehicle applications. In: 2011 IEEE 33rd International Telecommunications Energy Conference (INTELEC), pp. 1–9 (2011). doi:[10.1109/INTELEC.2011.6099803](https://doi.org/10.1109/INTELEC.2011.6099803)
25. Neema, H., Bapty, T., Batteh, J.: Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems. In: Proceedings of the 10th International Modelica Conference, pp. 235–245 (2014)
26. Newman, J., Thomas-Alyea, K.E.: *Electrochemical Systems*, 3rd edn. Wiley, Hoboken (2004)
27. Noll, C., Blochwitz, T.: Implementation of modelisar functional mock-up interfaces in SimulationX. In: 8th International Modelica Conference (2011)

28. Piller, S., Perrin, M., Jossen, A.: Methods for state-of-charge determination and their applications. *J. Power Sources* **96**(1), 113–120 (2001). In: Proceedings of the 22nd International Power Sources Symposium
29. Plett, G.L.: Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: part 2. Modeling and identification. *J. Power Sources* **134**(2), 262–276 (2004)
30. Pohlmann, U., Schäfer, W., Reddehase, H., Röckemann, J., Wagner, R.: Generating functional mockup units from software specifications. In: Proceedings of the 9th International MODELICA Conference, 3–5 September 2012, Munich, pp. 765–774 (2012). doi:[10.3384/ecp12076765](https://doi.org/10.3384/ecp12076765)
31. Puntigam, W.: Coupled simulation: key for a successful energy management. In: Virtual Vehicle 11th Automotive Technology Conference (2007)
32. Pussig, B., Denil, J., De Meulenaere, P., Vangheluwe, H.: Generation of functional mock-up units for co-simulation from simulink using explicit computational semantics. In: Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative, DEVS '14, pp. 38:1–38:6. Society for Computer Simulation International, San Diego (2014)
33. Qtronic: FMU SDK: free development kit (2014)
34. Swan, S.: An introduction to system level modeling in SystemC 2.0. *Review Literature and Arts of the Americas* (May), pp. 0–11 (2001)
35. SystemC Verification Standard Specification (2003)
36. Unterrieder, C., Huemer, M., Marsili, S.: SystemC-AMS-based design of a battery model for single and multi cell applications. In: 2012 8th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), pp. 1–4 (2012)

Languages, Design Methods, and Tools for Electronic
System Design

Selected Contributions from FDL 2015

Drechsler, R.; Wille, R. (Eds.)

2016, VIII, 193 p. 109 illus., 90 illus. in color., Hardcover

ISBN: 978-3-319-31722-9