FMI for physical models on automotive embedded targets

Christian Bertsch[1] Jonathan Neudorfer[1] Elmar Ahle[1]
Siva Sankar Arumugham[2] Karthikeyan Ramachandran[2] Andreas Thuy[3]

[1]Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, 71272 Renningen, Germany
[2]Robert Bosch Engineering and Business Solutions Private Limited, 560 103 Bangalore, India
[3] ETAS GmbH, 70469 Stuttgart, Germany

Christian.Bertsch@de.bosch.com Jonathan.Neudorfer@de.bosch.com Elmar.Ahle@de.bosch.com
SivaSankar.Arumugham@in.bosch.com Karthikeyan.R@in.bosch.com Andreas.Thuy@etas.com

Abstract

This paper explores the possibility to include and execute source code functional mockup units on Bosch electronic control units. A prototypical realization is presented, and assumptions as well as limitations are documented. Special emphasis is laid on requirements for the contained C-code. Furthermore, aspects for an extension to adapt the FMI to the usage on automotive embedded real-time systems are summarized.

Keywords: Automotive, FMI, Embedded Systems, Electronic Control Units

1 Introduction

The Functional Mock-up Interface (FMI) is a well received tool independent approach for model exchange (Blochwitz et al. 2011, 2012) and a promising candidate to become the industry standard for exchange of models and cross-company collaboration (Bertsch et al. 2014). From the beginning in the MODELISAR project, a wide range of possible simulation target platforms for FMI was foreseen, ranging from offline simulation platforms over Hardware-in-the-Loop (HiL) real-time systems to embedded systems (Chombard 2012). While in the offline simulation world FMI is well-established, this is not the case for embedded applications.

This paper presents results of a prototypical FMI implementation for physical models on automotive embedded targets. Furthermore, the usage and adaptation of the FMI as a standard for the automotive embedded world is investigated.

Section 2 provides an overview of the state of the art of physical models on real-time systems. In Section 3, a prototype implementation of FMI on an Electronic Control Unit (ECU) is introduced. Section 4 proposes aspects for an extension to adapt the FMI to the usage on automotive embedded real-time systems before Section 5 concludes with a summary and an outlook.

2 Physical models for real-time systems

First, an overview of real-time systems is given where physical models are already implemented using the FMI standard (Section 2.1). Then, the status on embedded systems is examined (Sections 2.2 and 2.3) and the idea as well as advantages of the FMI on embedded systems is presented in Section 2.4.

2.1 FMI for real-time systems: state of the art

The usage of FMI for real-time simulation is gaining importance. The import of Functional Mock-up Units (FMUs) is supported on major HiL systems in the automotive domain, e.g., ETAS LABCAR (Mitrohin 2014). The same is true for rapid prototyping applications (Brembeck et al. 2014). There are even first applications of FMI for productive online control and optimization tasks of power plants (Franke 2015). All of these real-time applications have one thing in common, i.e., they run on personal computers (PC) or similar systems.

2.2 Embedded systems and ECUs

In contrast to real-time applications running on PCs, software on automotive ECUs has special requirements. Some of them are more generally true for embedded systems, some of them apply only to automotive embedded systems such as special coding guidelines (MISRA 2013) or software architecture (AUTOSAR 2015).

Typically, a modern vehicle has a lot of different ECUs for different tasks such as a vehicle control unit, an electronic stability control unit, and last but not least an engine control unit for internal combustion engines. ECUs are embedded systems with a different design compared to other computer systems such as PCs: Usually they have restrictions on

- memory usage,
- power consumption as well as computational power,
- hard real time requirements, and

- available libraries: Not all mathematical functions known in the offline world, e.g., as declared in `math.h`, are available for embedded targets.

2.3 Physical models on ECUs: state of the art

Physical models play an important role in the advanced control of internal combustion engines. Application fields are, e.g., in the advanced control of the air system of internal combustion engines. Also the real-time capable implementation of advanced numerical methods such as implicit discretization of stiff ordinary differential equations for ECU software functions have been successfully implemented (Wagner et al. 2008) and are used in real-life applications. One major advantage of physical models is that they can reduce calibration effort and the number of characteristic maps (Seuling et al. 2012).

The demand for the implementation of physical models is rapidly growing due to the growing complexity of systems (e.g., hybrid electrical systems), higher demands on control and diagnosis due to efficiency and legislation. There is a need for new model-based methods for

- virtual sensors, i.e., observers,
- model-based diagnosis,
- inverse physical models as feed forward part of control structures, and
- model predictive control.

In many cases the solution of the physical equations can be separated from other of the control algorithms and thus this part could be encapsulated an FMU. An example for this is shown for the model-based diagnosis, as depicted in Figure 1.

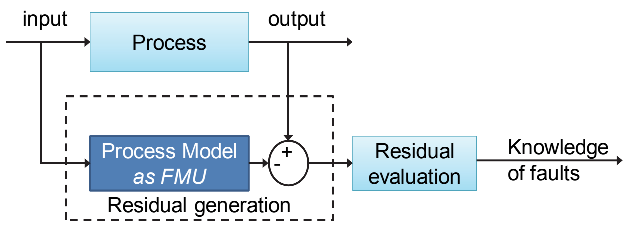

Figure 1: Encapsulated physical model as FMU in model–based diagnosis, adapted from (Ding 2013)

For the offline development and simulation of such physical model, powerful tools are available. However, the generated C-code from the majority of the tools cannot be ported directly to an ECU. Today, these models are often (re-)implemented individually for the specific use case on the ECU. This requires expert knowledge and significant effort.

Furthermore, there is no standardized interface for physical models available on the ECU. Although the AUTOSAR standard is widely accepted in the automotive domain, it is not supported by typical modeling tools for physical models. Also, AUTOSAR does not provide an interface for solvers of Ordinary Differential Equations (ODEs) and related mathematical description.

2.4 The idea and advantages of FMI on the ECU

The idea of using FMI not only in PC-based environments but also for embedded systems is already considered as a possibility in the existing FMI standard. Our motivation to work in this direction is to facilitate the implementation of physical models on the ECU by defining re-usable interface-functions and numerical algorithms. This enables improvement in the development process for physical models within ECU software: The models could be seamlessly reused from early design (offline), over rapid control prototyping, and then finally ported to the ECU. This will also facilitate the collaborative development of ECU software between original equipment manufacturers (OEMs) and their suppliers in the context of physical models.

There are a lot of tools available for physical modeling that support the FMI standard (also with C-code generation), but only very few that support the AUTOSAR standard. In a first step, one can apply the prototype presented below for rapid (control) prototyping purposes with physical models being exported as FMUs from different tools.

Later – when additional requirements on the included C-code would be fulfilled and the standard would be adapted – it could also be used to exchange code for real ECU projects. The core question is how to integrate the FMU and its numeric framework into an ECU software architecture like AUTOSAR.

3 Prototype implementation of FMI on an ECU

A first prototype for using FMI as an intermediate format in AUTOSAR was described in (Thiele et al. 2011) with the intention of using FMI as a lean standard to exchange software components. There it is pointed out that compared to AUTOSAR, the FMI standard is much smaller and more straightforward, and support of the FMI standard is a more manageable task. Additionally, (Thiele et al. 2011) describes in detail the mapping of FMI and AUTOSAR XML configuration files and interfaces. In our approach this is done in a similar way, as described in Section 3.2.1. However, in contrast to our intention, the emphasis of (Thiele et al. 2011) was on pure control software and on the software architecture. In this paper, the focus is laid on porting physical models to an ECU and actually computing them on the target platform.

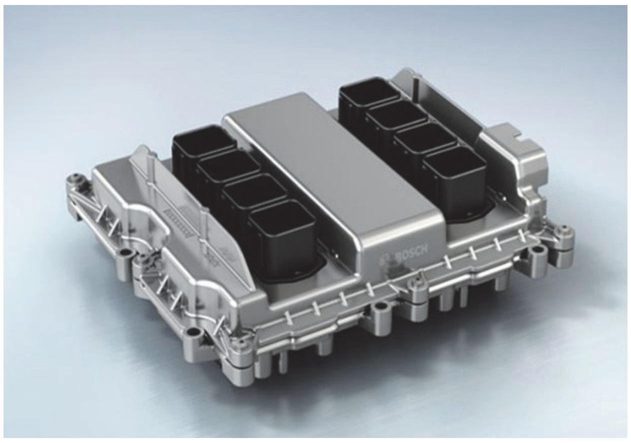
Figure 2: Bosch MDG1 ECU

As a demonstration platform the latest ECU platform of Bosch for powertrain applications is selected - the new “MDG1 ECU” as depicted in Figure 2. It is a scalable multi core processor system, see (Rüger et al. 2013) for details. As shown in Figure 3, this platform supports both application software (ASW) in a Bosch specific format compatible with former Bosch ECU generation “MEDC17”, and AUTOSAR. The base software is fully AUTOSAR compliant and communicates with the AUTOSAR application software via the run-time environment (RTE) and with MEDC17 application software with a Bosch-specific interface.

This is the platform for which the execution of FMUs has been prototypically realized within the real ECU software infrastructure (i.e., the build toolchain and special compilers).

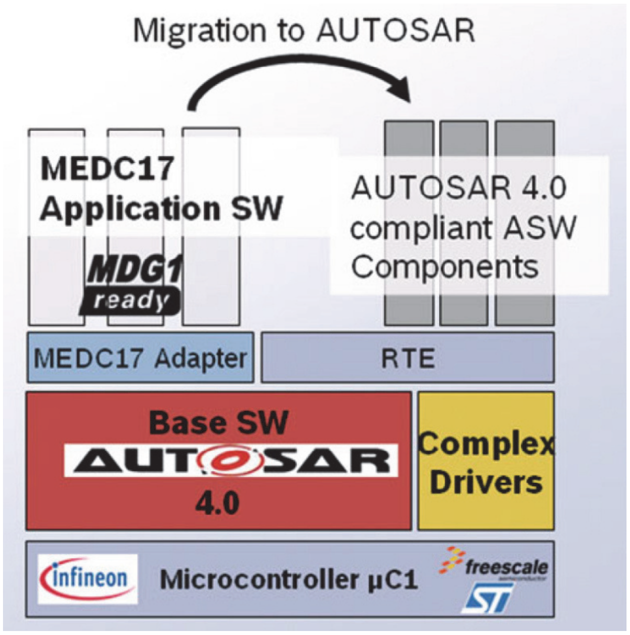

Figure 3: Software architecture of MDG1 ECU, (Rüger et al. 2013)

3.1 Considerations for the prototypical porting of FMUs to the ECU

The prototype is intended to work with source code FMUs from publicly available modeling tools. These tools were selected by inspection of the generated source C-code, w.r.t. the criteria listed in Section 4.2. As even the most suitable C-code did not fulfill all our requirements, modifications of the C-code are necessary. For this purpose a conversion tool was developed to perform this semi-automatically.

When considering the usage on FMI for embedded software, one has to choose between model exchange or co-simulation FMUs. Both types of FMUs have their pros and cons: co-simulation is closer to task-based execution in ECU software, while model exchange can interface special re-usable numerical algorithms optimized for the ECU that can be provided within a solver library.

The choice of the FMU type also depends on what the modeling tool can export: Some tools export very good variable step solvers for co-simulation FMUs intended for the PC-world but with no chance to compile and use such solvers for an embedded target. More suited for embedded targets are inline integration methods (Elmqvist et al. 1995).

On the other hand, model-exchange FMUs can take advantage of optimized solvers and numerics for the target platform without the above mentioned target dependency of the FMU. For this reason both, co-simulation and model exchange FMUs, are selected. However, regarding real-time capability, especially for model exchange FMUs, the prototype only supports a subset of the FMI standard necessary for the description of continuous ODEs neglecting events. The assumptions and limitations of the implementation are described in Section 3.4.

3.2 A prototypical workflow to bring FMUs on Bosch ECUs

This prototypical workflow addresses the transformation that an FMU must go through to be a part of an ECU software component, by utilizing the available ECU resources and numerical capability.
The whole conversion process can be done in a preparation phase offline on a development platform (typically a PC), where

- the FMU is unzipped,
- the `modelDescription.xml` file is parsed and mapped to the corresponding configuration XML file on the ECU software side, and
- the C-code is manipulated so that it can be compiled for the target ECU within the standard ECU software architecture before finally
- the manipulated C-code is included in an application software component that can be included in the overall ECU software.

Then, the software can be compiled and flashed to the ECU and then executed and validated.

3.2.1 Mapping of configuration files and interface definitions

The data description file of the FMI standard `modelDescription.xml` must be converted to the data description format for ECUs (e.g., ARXML in AUTOSAR). This is performed analogously to the way described by (Thiele et al. 2011). This task is done by the first stage of a script-based conversion tool (see Figure 4).

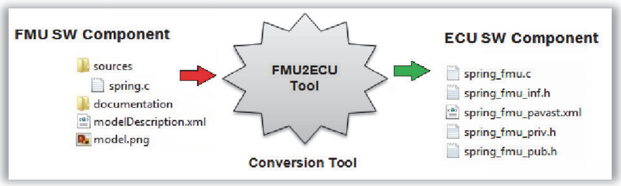

Figure 4: FMU file artifacts converted automatically into an ECU compatible software component

3.2.2 Conversion of C-source code from FMUs to ECU software C-code

Some changes like the selection of suitable data types (e.g., single precision float for `fmi2Real`) can be configured in the platform dependent configuration files `fmi2TypesPlatform.h` and `fmi2Functions.h`.

For other changes the second stage of the tool-supported conversion process applies: the C-code contained in the source code FMUs is converted to C-code that can be compiled for an ECU with the standard toolchain for Bosch engine control software. Since the FMU source code is generated from simulation tools typically operating in a PC environment, there are some code adaptations involved in the process of running them on ECU. However, the goal is to keep the code change as low as possible. Some examples of the changes of the C-code are:

- All macro definitions in `fmu.c` must be moved to private headers to avoid multiple definitions while compiling multiple FMUs together. For example, definitions such as the following should be put in private headers:
  ```
  #define NUMBER_OF_REALS XX
  #define NUMBER_OF_INTEGERS YY
  ```
- Information from FMUs that must be exposed to the outside shall be defined in public header files, e.g., reference to the FMU interfaces or function declaration prototypes such as
  `void modelName_fmi2instantiate(…);`
- Some header inclusions generated in FMU source files like `stdio.h`, `math.h` etc., must be removed or excluded since they are not available on the embedded platform.
- Mathematical function calls must be mapped to the available functions from the AUTOSAR base software.
- Floating point variables and constants within the code have to be converted to single precision. A notable compiler specific change is that for any arithmetic expression in model equations where float constants are used, they must be suffixed by `f` to enable the compiler to identify that it is a single precision float variable and not double precision. This means that the original code
  `mu*((1.0-x0*x0)*x1)-x0;`
  is replaced by
  `mu*((1.0`**`f`**`-x0*x0)*x1)-x0;`
- Implicit type castings are replaced by explicit type castings.
- Any explicit print functions like `fprintf()`, `printf()` should be removed.

The structure of the C-code contained in source code FMUs is very specific to the exporting tool. At the moment FMUs generated from three different tools are supported. The conversion of the C-code is performed mainly automatically, but still the inspection of the modified C-code and some manual adaptations are necessary.

3.2.3 Modification of memory allocation

On the ECU, dynamic memory allocation is not allowed. Instead, the memory demand must be known at compile time. Therefore, the required memory must be pre-allocated and later assigned via the FMI callback function `allocateMemory()`. However, it is not sufficient for the ECU implementation of the callback function to return just a pointer to any free space in memory. Instead, memory for each variable is pre-allocated individually and the correct memory location for each allocation request must be returned. This is important for debugging and calibration. The respective tools must be able to map the variable names to the resp. space in memory. For instance, a model might use a `struct` containing the parameters *a*, *b*, and *c*. Here, the calibration tool must be able to unambiguously map the variables *a*, *b*, and *c* to their respective locations in memory.

3.2.4 Concept of FMU computation algorithm

The modified C-code is called by the "FMU computation algorithm" which takes the role of the solver in the case of model exchange FMUs and the master algorithm in the case of co-simulation FMUs. This FMU computation algorithm is the application software component that can communicate with the other parts of the ECU software and is scheduled by the operating system.

The FMU computation algorithm can make use of solver libraries in the case of model exchange. It interacts with the modified C-code from the FMUs via the standard FMI interface functions according to the supported part of the FMI standard calling sequence.

This design enables easy integration of FMU components as ECU components, thereby enabling interfacing with the FMU computation algorithm, solver libraries, and other ECU components. A graphical representation of the FMU software components' organization inside the ECU is shown in Figure 5.

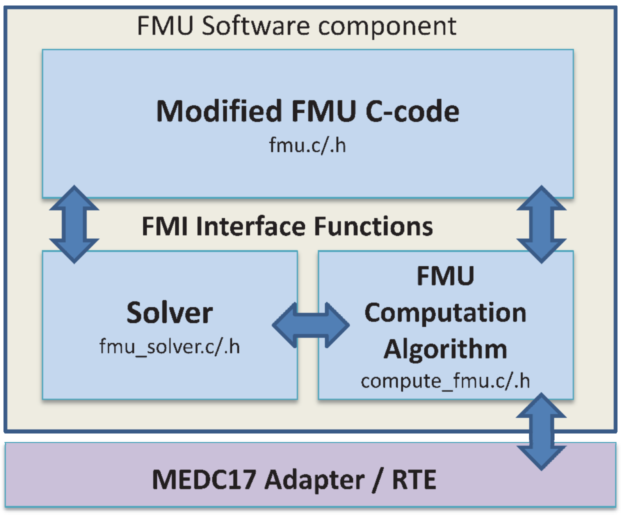

Figure 5: FMU software component for model exchange FMUs

As shown in the depicted setup, the FMU is accessed via the FMI interface functions, thus keeping the FMU's C-code with minimal changes while executing it on the ECU. The FMU computation algorithm is a combined set of functions defined, organized and distributed across different source files under a common container.

3.2.5 Example: double mass spring damper model

Several FMUs have been ported to the Bosch MDG1 ECU with different properties of the physical models such as number of states (up to >10), stiffness of the included ODEs and physical domains covered. The prototype workflow allows bringing physical models very easily on the ECU in a fraction of the time necessary to hand-code the model and solvers from scratch.

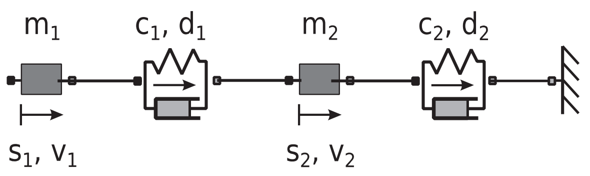

Figure 6: Schematic representation of the double mass spring damper model

To demonstrate the execution of FMUs running on an ECU, a stiff model of a double mass spring damper as depicted in Figure 6 is considered. This example serves as a benchmark problem for a stiff powertrain model that could be used in applications listed in Section 2.3. For this simple model, source code FMUs were generated with several Modelica-based simulation tools. The size of the contained C-code differed from 54kByte to 2.9MByte, i.e., differing by a factor of more than 50. This correlated also with the complexity and "readability" of the contained C-code and is a first indication, how feasible it is to re-use this C-code on embedded targets.

For the above model, the system of equations is given by

$$\begin{aligned} \dot{s_1} &= v_1 \\ m_1\dot{v}_1 &= c_1(s_2 - s_1) + d_1(v_2 - v_1) \\ \dot{s_2} &= v_2 \\ m_2\dot{v}_2 &= c_1(s_1 - s_2) + d_1(v_1 - v_2) - c_2 s_2 - d_1 v_2 \end{aligned} \qquad (1)$$

with the following parameters:

m_1=m_2=1kg, c_1=c_2=1N/m, d_1=100Ns/m, d_2=1Ns/m

and initial conditions:

s_1=2m, v_1=0m/s, s_2=2m, v_2=0m/s.

The solution calculated with a linear-implicit Euler solver with a step size of 10 ms demonstrates the expected damped oscillation of the masses. An explicit Euler solver is unstable for the same step size, as shown in Figure 7.

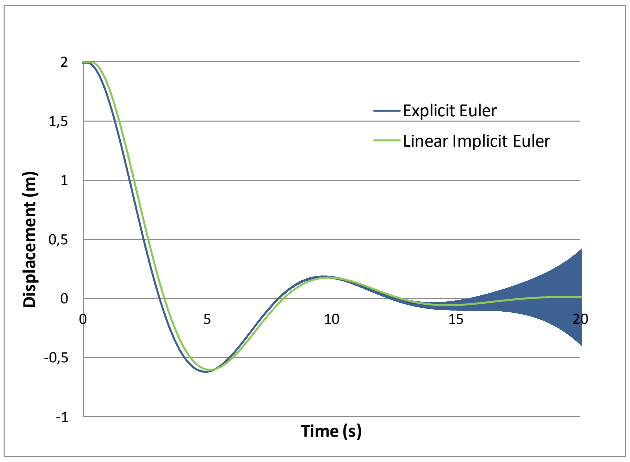

Figure 7: Displacement of mass m_2 provided by explicit and linear implicit Euler methods

This well know behavior of explicit solvers for stiff ODEs with large time steps also demonstrates the benefit of FMI for porting physical models on the ECU: (linear-)implicit solvers can be easily realized either by:

- Co-simulation FMUs generated by exporting tools supporting inline integration for implicit discretization methods of ODEs (Elmqvist et al. 1995)
- Model-exchange FMUs providing a standardized interface between the model equations (even with the possibility to calculate the Jacobian matrix in FMI 2.0) and an optimized implementation of numerical algorithms for the target platform.

In this example, a model exchange FMU generated from a commercial tool is used as well as our own implementation of explicit and linear-implicit Euler solvers on the Bosch MDG1 ECU.

3.3 Validation of FMUs running on the ECU

The computation results of the FMUs ported to the MDG1 ECU are validated in several steps:

- Creating reference signals by simulating the FMU offline on a PC. The FMU can also be embedded in a closed loop system model to derive meaningful stimuli for the further tests.
- Compiling the modified FMU within an ECU software with the Bosch standard toolchain and creating a Windows DLL that can be tested within the ETAS test tool RT2.
- Compiling the modified FMU within an ECU software with the Bosch standard toolchain for the target processor and running it on virtual hardware (Leupers et al. 2012).
- Running the FMU on the real target platform (Bosch ECU test board or series product). Stimulate and measure with an ETAS LABCAR Hardware-in-the-Loop (HiL) setup or a measurement and calibration tool such as ETAS INCA.

3.4 Restrictions of the prototype implementation

There are also some restrictions in the prototypical implementation of FMU like no event handling for model exchange and no variable step size for solvers. Otherwise one would have to take additional measures to guarantee real-time behavior and to map calculations to ECU tasks. Special assumptions on the contained C-code are made and adapted to the prototype to cope with FMUs generated from three different tools (including Modelica-based tools) which address the desire of using FMUs from commercial tools. However, C-code from many commercial tools is not suitable to run on an ECU due to their size and complexity since it was not intended to run on an embedded system.

At the moment, the prototype does not yet fulfill all requirements, e.g., regarding compliance with the software development guidelines for the C programming language by the Motor Industry Software Reliability Association (MISRA 2013), for series engine control software.

With the successful prototypical implementation, the next steps will be:

- Usage of FMUs for the computation of physical models as virtual sensors or in advanced control and diagnosis tasks as described in Section 2.3.
- Extension of the offline preprocessing and calculation algorithm for connected FMUs as well as the connection to other software components
- Full AUTOSAR support

4 Key findings and outlook

Based on the sample FMUs implemented on a Bosch ECU, the key findings are summarized and an outlook is given.

4.1 Aspects for embedded systems already contained in the FMI standard

In the original development of the FMI, some consideration was already given to the usage of FMI on embedded systems (cf. the FMI 2.0 standard), resulting in the following features which are required by such systems:

- Source code FMUs come with C-code, the most common language to program embedded systems.
- It is easy to replace data types (double precision float to single precision float).
- The interface description in an XML file can be mapped easily to corresponding XML files for ECU software components.
- Co-simulation FMUs with fixed communication step size can be mapped easily to ECU tasks with periodic activation
- Callback functions are defined for memory allocation and logging, which can be replaced or disabled by special mechanisms for the target platform.

4.2 Key findings: changes to source code FMUs for embedded targets

On ECUs, one is confronted with computation limitations compared to the offline world or the real-time PC environment. These limitations should be reflected in an FMU that is suitable for applications running on an ECU.

- Limitation 1: **Memory**
  The FMI purposefully leaves the organization of a model's data (e.g., parameters, internal variables) to the FMU in order to achieve maximum freedom. In contrast, ECU software is organized with respect to memory to allow transparency, simplicity, and efficiency. That means, if data structures are left to the freedom of the implementer, they still have to be transparent to the outside at least so far as to allow

parameterization and signal observation. Currently, this is not possible with the FMI since parameters, states, inputs, and outputs are not exposed directly to the outside but hidden behind access functions.

- Limitation 2: **Event handling**
  In general, events could increase the runtime for real-time systems in an unpredictable manner. There may be any number of events within a second which all trigger their respective event handling algorithms. Thus, in order to guarantee that models with event handling satisfy real-time requirements one will have to take extra measures.
- Limitation 3: **Algebraic loops**
  Similar to event handling, algebraic loops generally have an unpredictable impact on the run time of an FMU where they require an iterative solution. This restriction applies both to connected FMUs and to iterative solution methods within one FMU.
- Limitation 4: **User-interaction is impossible**
  Several features which make sense for offline simulations are either overhead or even dangerous for computations on the ECU. Such features which are either supported or not explicitly forbidden in the FMI include logging and I/O operations such as `print()`.
- Limitation 5: **Support functions**
  Support functions such as available from the `math.h` library are usually not available on the ECU. State of the art ECUs provide their own set of support functions through an AUTOSAR library. However, function names and usage of these are generally different from their offline counterparts.
- Limitation 6: **C-code compliance**
  Strict coding guidelines apply for source code that is executed on an ECU. Such requirements are standardized by the Motor Industry Software Reliability Association (MISRA 2013).
- Limitation 7: **Cross-compilation and object code FMUs**
  As ECUs are quite special platforms, they require special compilers and build processes as well. Object file FMUs have to be cross-compiled accordingly which requires the availability of the suitable toolchain for the target platform
- Limitation 8: **Available data types**
  In order to provide optimized code, the set of available data types should be enlarged. For example, one should be able to distinguish a `uint8` from a `uint32` variable.

Beyond the mentioned restrictions, to achieve a convincing performance of FMUs running on an ECU, careful consideration of the target platform is required:

- The solver must be able to make use of the platform's computing capabilities, which differ widely from one platform to another.
- Computations are mostly performed using single precision floats.
- Heap and stack usage must be minimized.

4.3 Outlook towards a future FMI variant for embedded systems

The limitations listed in the previous section will have to be addressed by a possible future FMI standard. Major changes to the existing FMI 2.0 standard will be necessary that it does not seem to be possible to include them very easily, as there will be not only requirements on the interface such as for FMI today, but also on the contained C-code. Thus, a variant of the FMI standard especially for automotive embedded targets and a connection to the AUTOSAR standard is desirable. FMUs generated according to such a standard could then be executed on ECUs without the conversion steps presented in Section 3 of this paper. Such FMUs could really be seamlessly used for all cycles of the development process until running on automotive embedded systems.

5 Summary

In this paper, it is shown by a prototype that the current FMI standard with some modifications which are highly tool-dependent allows the computation of FMUs of physical models on an ECU (as a representative for embedded controllers) as long as the FMU satisfies some assumptions and limitations. This was demonstrated for FMUs generated from different tools on a Bosch MDG1 engine control unit, where the modified FMUs have been included in the real software build process. The prototype can be used for rapid control prototyping with physical models.

For usage of FMUs out of the box in productive ECU software, the standard will have to be modified, mostly enforcing the above mentioned restrictions (Section 4.2). ECU software must be lean, efficient, and above all, safe. We foresee the benefits for the establishment of an "FMI for automotive embedded systems" for seamless model-based design until the execution on the target platform.

Acknowledgements

We would like to thank Martin Otter, Andreas Pfeiffer, and Jonathan Brembeck from DLR as well as our colleagues Andrea Flexeder, Eckart Mayer-John, Matthews Peter, Dibakar Mahalanabish, Naresh Mandipalli, Wolfgang Lengerer, and Marcus Bossler for fruitful discussions.

References

AUTOSAR Consortium, Autosar Standard 4.2, available at http://www.autosar.org/, 2015

Bertsch, C., Ahle, E., Schulmeister, U., The Functional Mockup Interface - seen from an industrial perspective, In: Proceedings of the 10th International Modelica Conference 2014, Lund, Sweden

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: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany

Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauß, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A., The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models, In: Proceedings of the 9th Modelica Conference 2012, Munich, Germany

Brembeck, J., Pfeiffer, A., Fleps-Dezasse, M., Otter, M., Wernersson, K., Elmqvist, H., Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface, In: Proceedings of the 10th International Modelica Conference 2014, Lund, Sweden

Chombard, P., Multidisciplinary modeling and simulation speeds development of automotive systems and software, ITEA2 innovation report, 2012, published online: https://itea3.org/project/modelisar.html

Ding, S. X., Model-Based Fault Diagnosis Techniques, Springer, 2nd edition, London 2013

Elmqvist, H., Otter M., Cellier, F.E.: Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems, In: Proceedings of ESM'95, European Simulation Multiconference, 1995

Franke, R., Mathematical optimization of dynamic systems with OpenModelica, Annual OpenModelica Workshop 2015, published online: https://openmodelica.org/images/docs/openmodelica2015/OpenModelica2015-talk02-Franke_Optimization.pdf

Leupers, R., Martin, G., Plyaskin, R., Herkersdorf, A., Schirrmeister, F., Kogel, T., Vaupel, M., Virtual Platforms: Breaking New Grounds, IEEE DATE Conference, Dresden 2012

Mitrohin, C., FMI in LABCAR HiL; From MiL to SiL towards HiL, FMI-Tutorial, 10th International Modelica Conference 2014, Lund, Sweden

MISRA Consortium, MISRA C: 2012 Guidelines for the use of the C language in critical systems, 2013, available from http://www.misra.org.uk

Rüger, J.-J., Wernet, A., Kececi, H.-F., Thiel, T., MDG1: The New, Scalable, and Powerful ECU Platform from Bosch, Proceedings of the FISITA 2012 World Automotive Congress - Volume 6: Vehicle Electronics, Springer, 2014

Seuling, S., Hamedovic, H., Fischer, W., and Schuerg, F., Model Based Engine Speed Evaluation for Single-Cylinder Engine Control, SAE Technical Paper 2012-32-0044, 2012

Thiele, B.; Henriksson, D., Using the Functional Mockup Interface as an Intermediate Format in AUTOSAR Software Component Development, In: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany

Wagner, A., Bleile, T, Lux, S., Fleck, C., Method for real time capability simulation of an air system model of an internal combustion engine, US Patent US 8321172 B2, 2008