ITEA Office
High Tech Campus 69 - 3
5656 AG Eindhoven
The Netherlands

T + 31 88 003 6136
E info@itea3.org
W www.itea3.org

ITEA 3 is a EUREKA strategic ICT cluster programme

EMPHYSIS – D7.9 *eFMI for physics-based ECU controllers Public report*

ITEA 3

Version 1, September 2021

Contributors (in alphabetical order):

AbsInt Angewandte Informatik GmbH
Dassault Systèmes AB
Dassault Systèmes SE
DLR Institute of System Dynamics and Control (task leader)
dSPACE GmbH
EFS GmbH
ESI ITI GmbH
ETAS GmbH
GIPSA-lab/Grenoble INP
Maplesoft
Mercedes-Benz AG
Renault SAS
PikeTec GmbH
Robert Bosch GmbH
Siemens Digital Industries Software
Siemens NV
Volvo Car Corporation

Project Acronyms

AlgCL	Algorithm Code Language → GALEC
AlgCode	Algorithm Code
ARXML	Classic AUTOSAR interface description file
BinCode	Binary Code
BMI	Bosch Maintainability Index
eFMI	FMI for embedded systems
eFMU	Embedded Functional Mock-Up unit
EqCL	Equation Code Language
EqCode	Equation Code
FMI	Functional Mock-Up interface
FMU	Functional Mock-Up unit
GALEC	Guarded Algorithmic Language for Embedded Control
KPI	Key Performance Indicator
ProdCode	Production Code

eFMI Open Source Software (BSD 3-clause)

github.com/modelica/efmi-testcases	Modelica package with eFMI test cases for demonstrating and evaluating eFMI tooling (developed by Dassault Systèmes AB, DLR-SR, Robert Bosch GmbH)
github.com/modelica/efmi-containermanager	C# tool for creating, checking, reading and modifying eFMUs and their individual containers (developed by dSPACE GmbH)
github.com/modelica/efmi-compliancechecker	Python tool for checking eFMUs for conformance with the eFMI Standard (developed by ESI ITI GmbH)

Table of contents

1. Overview

The eFMI (FMI for embedded systems) standard has been developed in the ITEA3 project EMPHYSIS (2017 – 2021) [2]. It shall enable automated workflows from high-level mathematical models of physical systems (referred to as physical models) to embedded implementations on dedicated target environments, such as automotive electronic control units. As the key project result the eFMI specification has been published as eFMI 1.0.0-alpha.4 on https://emphysis.github.io/ [3]. A summary of the most important features and the relationship to FMI (Functional Mock-up Interface) is given in [1]. Core partners of the EMPHYSIS project have founded the Modelica Association Project eFMI (MAP eFMI - https://efmi-standard.org/) that is working on the first official release - version 1.0.0.[1]

Embedded software is commonly used on an embedded device to control or monitor a real-world system. The goal of eFMI is to support automated tool chains to utilize physical models in the software, in order to achieve a significantly better performance of the real-world system by using for example:

- *observers/virtual sensors* (for example extended and unscented Kalman filters, moving horizon estimation),
- *model-based diagnosis* (for example signal based fault detectors, linear/nonlinear residual generators),
- *feedback and feedforward controllers* (for example linear controllers with gain scheduling, nonlinear inverse models, nonlinear dynamic inversion, feedback linearization, linear/nonlinear model-predictive control),
- *neural networks* to approximate physical models and/or the above applications.

These types of functions are typically hand-coded software implemented and tested in an elaborate and time-consuming fashion. The eFMI standard aims to provide model exchange capabilities that allow to transfer physical models created in dedicated modeling and simulation tools to embedded code generating tools. This enables an end to end workflow from physical modeling to the deployment of the software function on an embedded device.

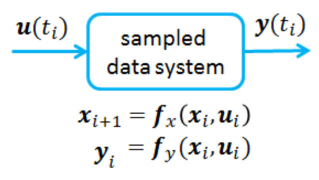


Figure 1: Mathematical description of eFMI component.

The eFMI workflow is illustrated in Figure 2. Starting point is an acausal or causal *Model* (for example a Modelica or AMESim model) that is transformed to a *sampled input/output block* with *one* (potentially varying) sample period for the whole block, see Figure 1. Such a block is defined as an eFMI *Algorithm Code* described by the Guarded Algorithmic Language for Embedded Control (GALEC), a target independent model representation with guarantees in terms of execution time, boundaries and exception free execution [1]. This representation is stored in the eFMI container architecture and can be further processed to *Production Code* for use on embedded targets.

The *Production Code* representation defines one or more mappings of an *Algorithm Code* representation to C or C++ Code (for example 32-bit and/or 64-bit representation of floating-point

[1] In this section, some text is used from eFMI 1.0.0-alpha.4 in a slightly modified form.

numbers, generic ANSI C-Code and/or code specialized to a particular target environment like AUTOSAR and/or specific target processors). The *Binary Code* representation provides one or more target specific executable codes for one *Production Code* representation. The *Behavioral Model* representation provides reference results to allow automatic verification of the *Production* and *Binary Code* representations.

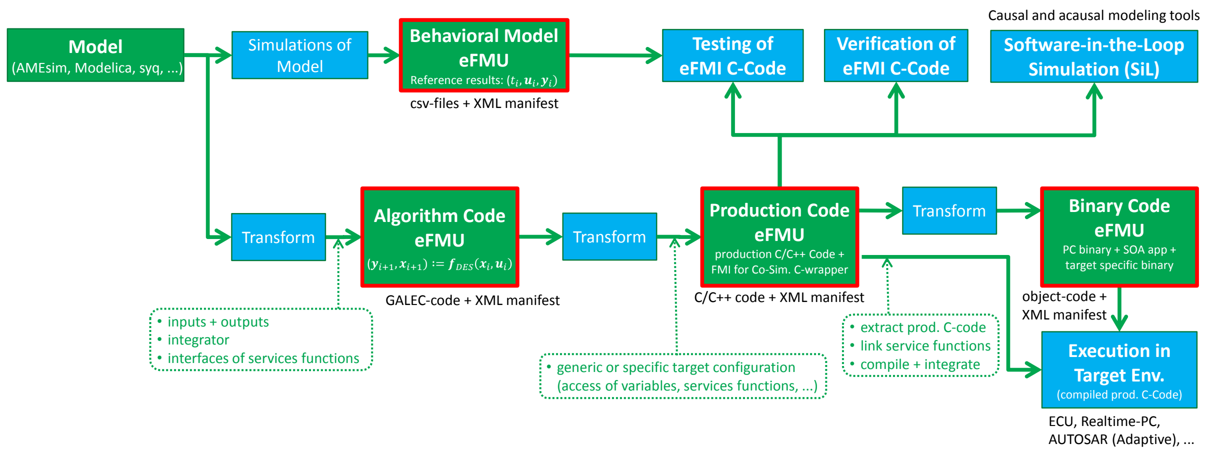

Model
(AMEsim, Modelica, syq, ...)
Simulations of Model
Behavioral Model eFMU
Reference results: (t_i, u_i, y_i)
csv-files + XML manifest
Testing of eFMI C-Code
Verification of eFMI C-Code
Causal and acausal modeling tools
Software-in-the-Loop Simulation (SiL)
Transform
Algorithm Code eFMU
$(y_{i+1}, x_{i+1}) := f_{DES}(x_i, u_i)$
GALEC-code + XML manifest
Transform
Production Code eFMU
production C/C++ Code + FMI for Co-Sim. C-wrapper
C/C++ code + XML manifest
Transform
Binary Code eFMU
PC binary + SOA app + target specific binary
object-code + XML manifest
Execution in Target Env.
(compiled prod. C-Code)
ECU, Realtime-PC, AUTOSAR (Adaptive), ...
• inputs + outputs
• integrator
• interfaces of services functions
• generic or specific target configuration (access of variables, services functions, ...)
• extract prod. C-code
• link service functions
• compile + integrate

Figure 2: eFMI workflow showing the different types of eFMI Model Representations (red boxes) stored in the eFMI Container Architecture.

A Functional Mock-up Unit for embedded systems (eFMU) can be packed in different formats. In particular, it can be packed as FMU and can then be simulated with any FMI-for-Co-Simulation compliant tool (see https://fmi-standard.org/tools) to perform Software-in-the-loop (SiL) testing. Code generation for an embedded device requires however dedicated tool support for eFMI. For more details, see eFMI 1.0.0-alpha.4.

The remaining part of this report summarizes the demonstrators that have been implemented during the EMPHYSIS project to develop the eFMI tools, proving the viability of the eFMI workflow and quantifying its benefits.

Sections 2.1 and 2.2 summarize the comprehensive tests, code quality assessments and runtime measurements on an ECU target to evaluate the technology readiness of the tool chain as well as the benefits in comparison with state-of-the-art development processes.

The demonstrators in sections 2.3 and 2.4 use the not-yet fully defined *Equation Code* model representation (not included in eFMI 1.0.0) which is in development for a future eFMI release. The *Equation Code* model is based on the not-yet fully defined *Flat Modelica Language* as an intermediate step from an acausal model to a causal GALEC model.

The demonstrators in sections 2.5, 2.6, 2.7 and 2.9 utilize the generation of neural networks to transform sophisticated non-real-time capable physical AMESim models into eFMI real-time models integrated with embedded software. The remaining demonstrators rely on a model transformation from acausal Modelica models to GALEC code and illustrate the usage of the technology in different controller and observer applications.

2. Demonstrators

2.1. D7.1 Demonstrator for test cases

In the development of the eFMI specification during the EMPHYSIS project an essential part was the continuous, parallel implementation of features and techniques in prototype tools. This approach showed issues in the design of eFMI in an early phase that could be discussed and fixed in several iterations of the specification process by inclusion of the important players (tool vendors and application specialists).

Both simple and more complicated benchmark examples have been utilized for testing. In total 48 test cases (including variants) have been implemented to be processed by the different eFMI tools involved. Most of the test cases are provided in the open source Modelica library eFMI_TestCases (https://github.com/modelica/efmi-testcases), some of the test cases are AMEsim models and some of them are manually implemented eFMUs. For each of the test cases reference results are provided in an eFMI *Behavioral Model* representation, that is automatically generated by scripts in Dymola using the eFMI Container Manager. The AMEsim models are exported by Simcenter Amesim and the Modelica models are exported by the Modelica tools Dymola and SimulationX to eFMI *Algorithm Code* representations. The tools support most of the partially very ambitious features requested by the test cases:

- Nonlinear inverse models
- Feedback-linearization based controllers
- Explicit and implicit integration schemes
- Event-based re-initialization of continuous states
- Neural networks
- Important built-in functions:
  - Solving linear equation systems
  - 1-D and 2-D interpolation of tables
- Error handling
- Implicit saturation

The initialization of Modelica models and their mapping to eFMI GALEC code turned out to be challenging, therefore this topic has to be investigated in more detail in future work. All 67 generated eFMI Algorithm code representations of the test cases are successfully imported by the Production code tools TargetLink, SCODE-CONGRA and ESP. The tools generate eFMI *Production Code* representations that contain a double precision floating-point (64 Bit) and a single precision floating-point (32 Bit) version of the C code. These representations also include self-containing implementations of all used built-in functions in the two architecture variants. With ESP there is also a first prototype that generates eFMI *Binary Code* by importing all the *Production Code* eFMUs of the test cases. By AUTOSAR Builder a few *Binary Code* eFMUs are generated dedicated for AUTOSAR Adaptive targets.

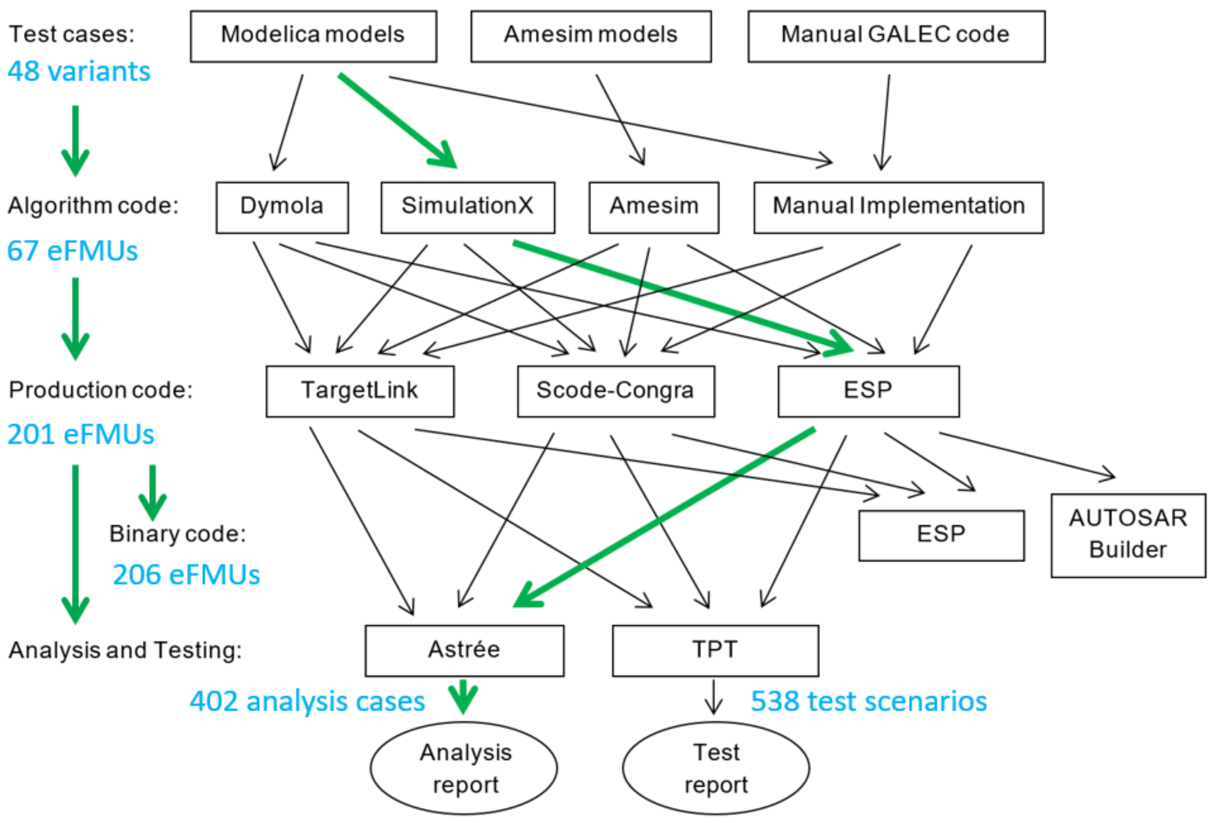


Figure 3: Variety of eFMI tool chain paths by the test case demonstrator tools; one possible path is highlighted.

All of the 201 Production code eFMUs (generated in 30 different tool chain paths, when considering 32 Bit and 64 Bit C code as separate paths) are successfully and automatically imported, analyzed and tested by the tools Astrée for static code analysis of 402 C code projects and TPT for 538 test run executions. In Figure 3 the variety of the tested tool chain paths for the test cases is illustrated.

The static code analysis of Astrée reports no or acceptable warnings for 90% of the tests, MISRA rule violations for 9% of the tests and definite errors for 1% of the tests. Most of the rule violations are detected in the implementation of complex built-in functions that were not in the focus of the project. The execution tests with TPT where simulation results are compared with the provided reference results are successfully passed by more than 96% of the runs. A wrong initialization of a variable in one test case is the single cause of all detected errors (by Astrée and TPT). It is planned to fix this issue and to further improve the compliance with MISRA rules after the EMPHYSIS project. Nevertheless, the very positive test rate impressively shows the maturity of the tool prototypes and their compatibility based on the alpha version of the eFMI standard.

2.2. D7.2 Demonstrator for eFMI performance assessment

2.2.1. Objective

The goal of the performance assessment is to benchmark code generated by the most common eFMI workflow against hand coded implementations according to state-of-the-art software development.

The evaluation criteria of this assessment are:

- Runtime performance on an Electronic Control Unit (ECU).
- Resource demand (code memory and data memory).
- Development effort.
- Code quality.

2.2.2. Procedure

The considered test cases of this performance benchmark [4] have been selected based on the input from experienced automotive software developers aiming to cover key challenges such as non-linearities, stiff dynamics, large number of states, large maps and advanced symbolic transformation from DAE to a suitable compact state space formulation.

The test cases and the corresponding major challenges are documented in Table 1. This set of performance benchmark test cases can be considered to represent a wide range of automotive applications.

The IDs refer to the test case number of the EMPHYSIS Test Cases Modelica library (D7.1) developed in the EMPHYSIS project and published together with the eFMI standard.

ID	Name	Difficulty	Major Challenge
M03	PID	Low	Minimal footprint incl. saturated IOs
M04	Drivetrain	Medium	Inverse linear physical model
M15	Air System	Medium	Stiff ODE with delay operator
M10	Inverse Slider Crank	High	Inverse non-linear physical model (DAE-Index 1)
M14	Rectifier	High	Advanced symbolic transformation to compact ODE form
M16	ROM	High	High dimensional maps, solve a large linear equation system

Table 1. All test cases (ordered by difficulty of the models) for an automated procedure to achieve the same quality as a manual implementation, and the represented major challenges.

The performance benchmark test cases have been implemented as Modelica models as starting point of the eFMI tool chain. All available *Algorithm Code* generating tools (SimulationX, Dymola) have been used to generate eFMUs considering all parameters as constant to achieve best computational performance. These eFMUs have then been extended with *Production Code* model representations using a compiler configuration for optimized computational performance.

The hand coded implementations have been derived from the literature of the defined mathematical problem or the equations of the Modelica model (they have *not* been derived from the generated Algorithm Codes).

The C code from the manual implementation respectively the code extracted from the eFMI *Production Code* model representation have been integrated with a test harness into a standard ECU software build. The compiled binaries have been executed on a Bosch Multicore ECU (MDG1).

The code is executed in an open loop fashion using the recorded stimulus signals as inputs retrieved from local memory. The run time (execution time) is captured based on the CPU ticks elapsed, right from the start of calling the model interface function (DoStep) until the function execution is completed. Hence, the overhead of the test harness has no impact on the measured runtime.

The results computed on the ECU have been verified against the simulated reference solutions considering the defined absolute and relative tolerances.

Furthermore, the development effort has been estimated for both eFMI and manual workflow. The relative gain of productivity ΔP has been determined by putting the total development efforts of the eFMI workflow $t_{dev,efmi}$ in relation to the state-of-the-art hand coded implementation $t_{dev,sota}$.

$$\frac{t_{dev,sota} - t_{dev,efmi}}{t_{dev,sota}} \quad (1)$$

The total development efforts are counted in working hours including the efforts for modeling, implementation and validation.

The code quality assessment for the performance benchmark test cases has been performed according to the Bosch regulations for hand coded production software. For some kinds of auto-generated code special rules have been defined but are not yet available for eFMI production code. Therefore, also the eFMI production code is assessed by the stricter rules for hand code.

2.2.3. Results Summary

The code for all test cases has passed validation and verification, except of the code for M10_A, where the tolerances have been exceeded. Nevertheless, did the code have a stable behavior. From the hand coded solution, it was known that a stable behavior was possible only by using a higher resolution for some of the state variables.

With respect to runtime performance (see Figure 4) all examples reached the target KPI (Key Performance Indicator) of less than 125%. In fact, in all test cases except of M16_A (considering the error handling overhead for M14_A), the auto-generated solution is even faster. The big deviation of M10_B is explained by the fact that the manual implementation required some double precision operation in order to achieve a stable result, which are very expensive on the ECU and lead to a more than two times longer execution time than the single precision variant.

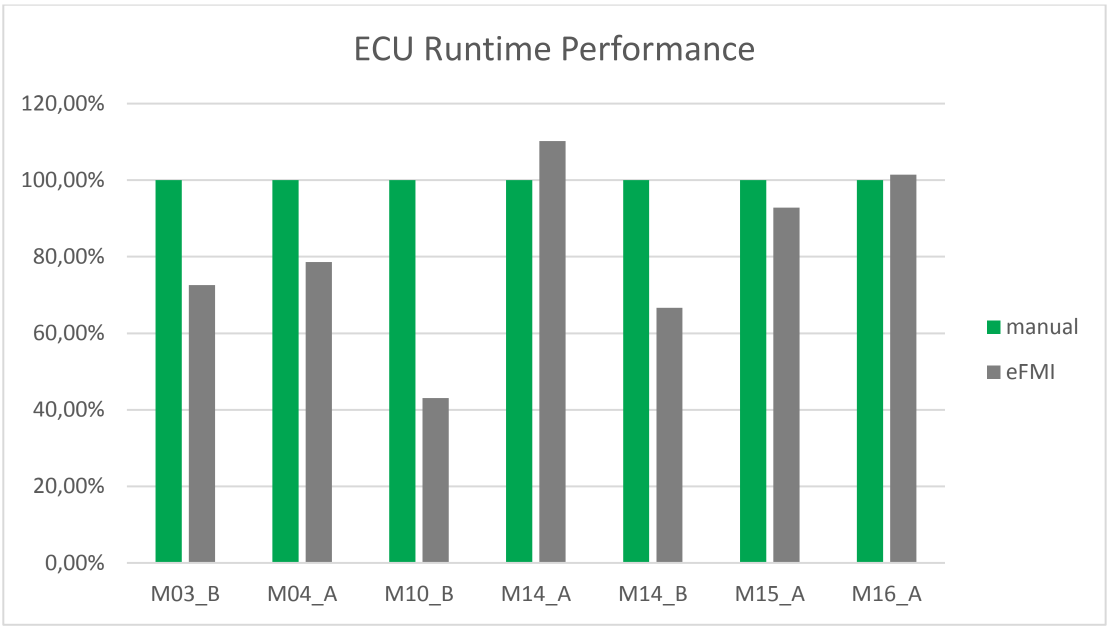


Figure 4. Results of the runtime measurements of all applications

The memory consumption summarizes the total code and data bytes that each model's source code consumes on the ECU. The code and data memory consumptions relative to the hand coded solution are shown in Figure 5 and Figure 6 respectively.

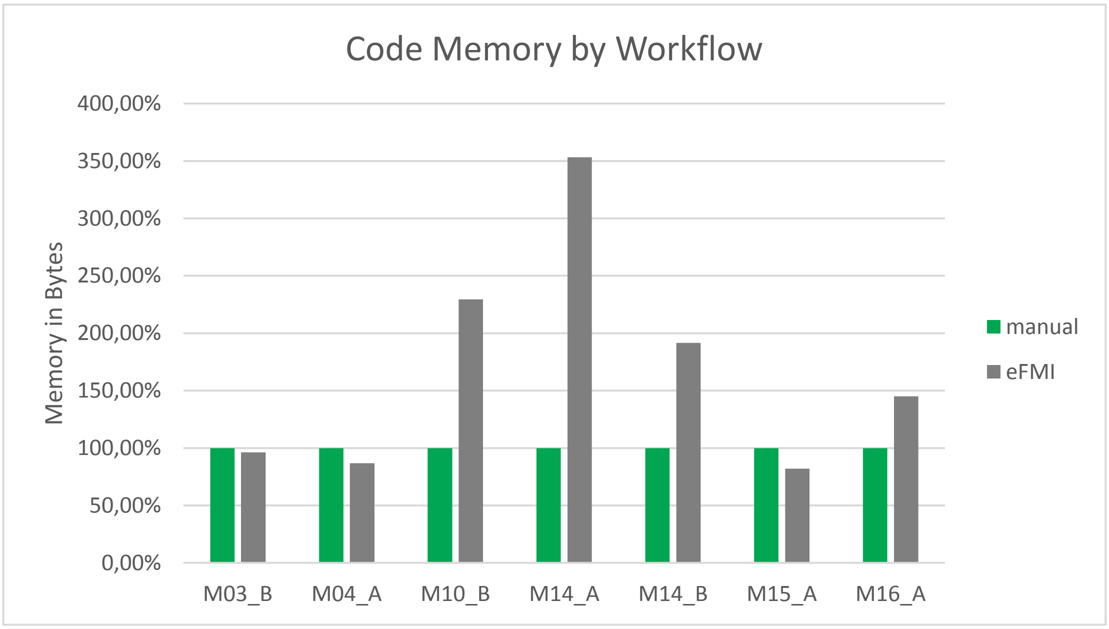


Figure 5. Results of the code memory in bytes for all applications

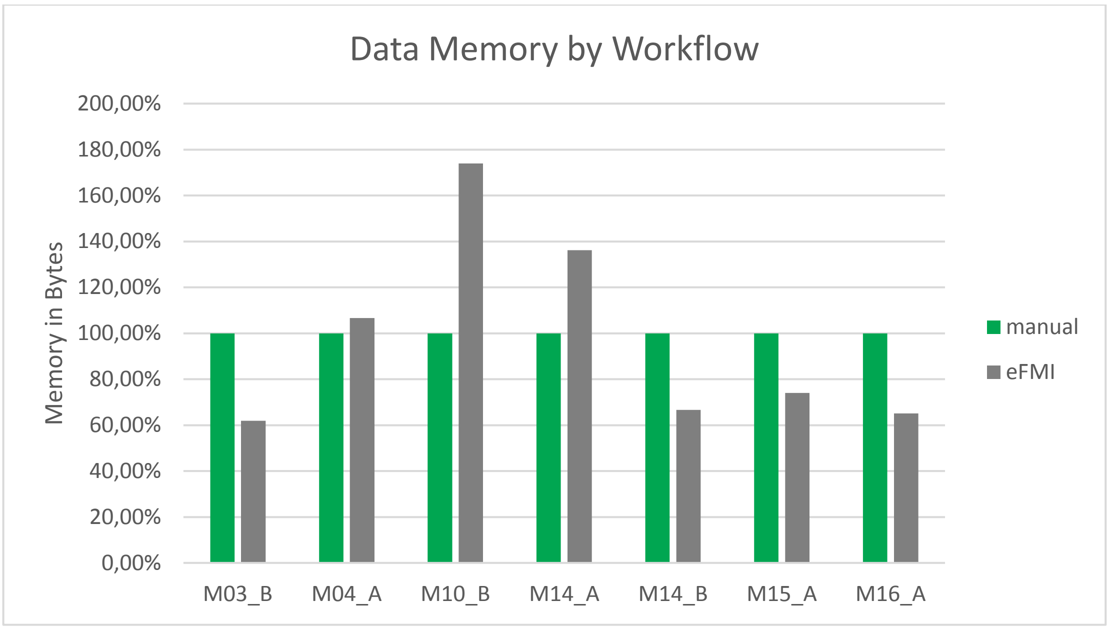


Figure 6. Results of the data memory in bytes of all applications

With respect to the memory demand a similar conclusion as for the runtime performance can be drawn for M03, M04 and M15 reaching the target KPI or outperforming the hand coded solution.

A deviation can be observed for M10. In this case both the code memory (+230%) and data memory (+170%) of the auto-generated code are significantly larger. This can be explained by the slider crank mechanism being described in spatial coordinates based on the Modelica.Mechanics.Multibody library. More suitable for planar mechanisms is the PlanarMechanics library (https:/github.com/dzimmer/PlanarMechanics). This will most likely lead to a more compact formulation which should result in less code and less data memory.

A similar remark relates to the rectifier test case M14. The auto-generated code of M14_A requires significantly more code memory (+350%) and more data memory (+135%) than the hand coded solution. This can be explained by the chosen hard problem where it was known that a very compact solution exists that can be written as a single line differential equation. This compact solution was not found by the symbolic processing of the equations derived from the component-oriented Modelica model. The case M14_B is an alternative implementation in Modelica code where the same prior knowledge is used in the Modelica implementation of the model. This compact Modelica model is then giving good results with more code memory but less data memory.

The results of the effort estimations are illustrated as bar diagram in Figure 7. The examples M03, M04 and M10 all show a productivity gain of ~90% compared to the state-of-the-art. In these cases, it is not only the implementation and validation effort that has been significantly reduced, but also the modeling effort is much smaller. This is due to the fact that in these cases a component-oriented approach has been used to graphically build-up the model from existing component models. Especially to manually derive the equations of motion for the slider crank mechanism of the test case M10 with a closed kinematic loop took not only a lot more time but also required the rare skills of an expert to find a proper solution of the inverse problem. These examples demonstrate how the modeling effort can be reduced from month to weeks or days for comparatively simple examples like a PID controller as well as complicated physical systems.

The rectifier example M14 can be seen as a counter example where for a small set of equations an easy to implement and very compact formulation of the problem is already known. In this rare case the direct implementation can be faster even though this assumption may not hold as soon as this model becomes part of a larger set of equations.

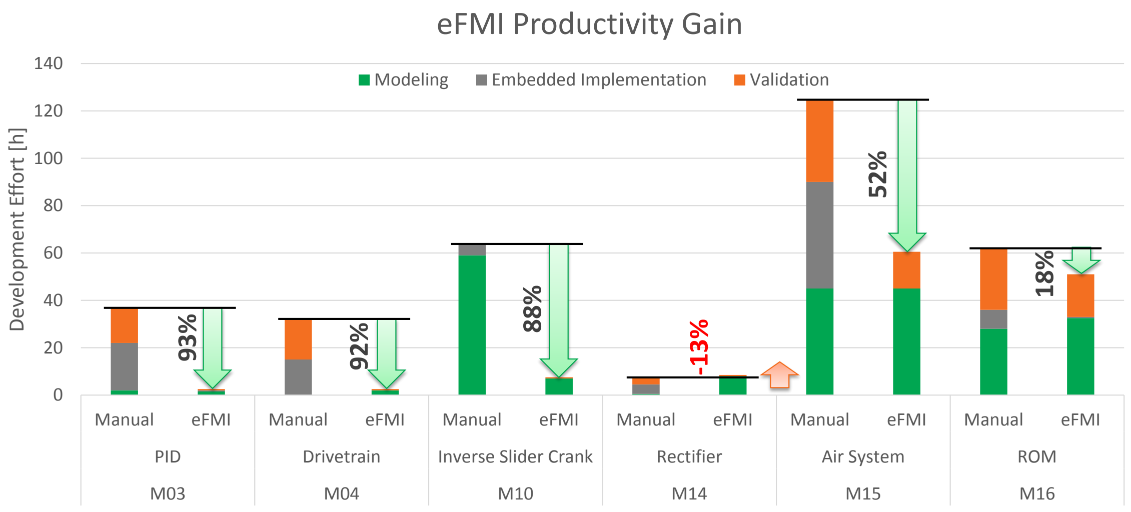


Figure 7. Results of the productivity gain of all considered applications.

In terms of the code quality assessment only the two test cases M04 and M15 have been considered due to the large effort of checking all available tool variants. A summary of the result of the static code analysis performed with Bosch Static Code Checker (Software Analyze, MISRA C: 2012) including the Bosch Maintainability Index (BMI) is provided in Table 2.

Table 2 Preliminary results of Static Code Check and Bosch Maintainability Index summary of eFMUs generated by tools (best result in bold).

ID	Test Case Name	Compiler Warnings Min./ Max. / Avg.	MISRA Findings Min./ Max. / Avg.	BMI Min./ Max. / Avg.
M04	Drivetrain	**0** / 14 / 8	**44** / 219 / 107	46 / **82** / 64
M15	Air System	**2** / 15 / 9	**41** / 250 / 112	33 / **75** / 58

The MISRA and compiler warnings for the eFMUs are mostly reported from the support source files (math files) shipped with the eFMUs. In order to ensure that the SW components pass the quality gates of series software production, their compiler warnings and MISRA findings must be solved, and for warnings / MISRA findings that shall not be solved, relevant technical details must be provided in the assessment report. The findings have been reported and discussed with the tool vendors and further improvements have already been implemented in the tool prototypes. Hence, the reported figures should be seen as preliminary results as the latest changes are not yet considered.

The Bosch Maintainability Index (BMI) is computed based on the top-level file artifact which defines the standard eFMU functions (Eg: Initialize/ DoStep), and the most complex among those functions is taken, usually DoStep(). The pass criteria for BMI is usually >-50 and within the range of -100 to 100, the higher the value of BMI, better the quality.

With respect to the BMI it can be summarized that the best generated solution is better than the hand coded solution. This leads to the conclusion that the generated code is not necessarily more complicated so that in case of code inspection no additional burden has to be expected by using eFMI. In terms of maintainability (an inherently complex problem), it can be stated that a physical model is significantly easier to maintain than a manually derived and hand coded algorithm.

2.2.4. Summary and conclusions

The presented performance benchmark examples illustrate that a significant gain of ~90% productivity in model-based development of physics-based control functions can be achieved. In terms of computational performance and memory demand, compared with hand coded solutions, no compromises have to be made. Against the expectation before project start, the highly optimized auto-generated solutions were able to outperform the hand coded solutions in most cases.

Even though counter examples may exist where an automated symbolic transformation is not able to find the same compact formulation as an experienced user, it can be summarized that with growing complexity of the model the benefits of the proposed eFMI workflow in terms of productivity and performance are increasing.

With respect to code quality short comings w.r.t. MISRA compliance, especially in the provided mathematical service functions, have been revealed. This will have to be improved in the future, but there is no indication that this could be a serious obstacle.

In summary the technology enabled by the eFMI standard can be seen as a breakthrough for physics-based functions for embedded software. This becomes even more apparent considering examples where due to the complexity of the tasks a manual implementation would not be commercially viable.

As an outlook it can be stated that very promising approaches using higher order real-time solvers such as the Rosenbrock method are expected to be fully functioning soon after the end of the project. This will allow to realize much smaller sampling rates for numerically challenging examples. In general, it must be emphasized that the evaluated tools are in a prototype stage. An even better performance can be expected when these tools have grown into mature products.

2.3. D7.3 Demonstrator for power train vibration reduction

2.3.1. Objective

The goal of this demonstrator is to investigate the eFMI workflow incorporating the *Equation Code* Model representation according to Figure 8 applied to a physics-based controller for the reduction of drivetrain vibrations.

The non-linear physical model of the drivetrain shall be used to design a feed-forward controller based on a simplified inverted plant model. The standard path to derive this kind of physics-based controller is part of the eFMI test cases described in D7.1. In this demonstrator the alternative path is explored. This includes the usage of multiple eFMUs with model representations as *Equation Code*, *Algorithm Code* and *Production Code*. In contrast to D7.1 is the *Production Code* generated as AUTOSAR application software component (SW-C).

The OpenModelica Flat Modelica export prototype and corresponding import prototype of SCODE-CONGRA are demonstrated. It is illustrated how the non-Modelica tool SCODE-CONGRA can be used to derive an inverse plant model from an acausal Modelica model.

The generated SW-C is finally tested on a Bosch MDG1 ECU.

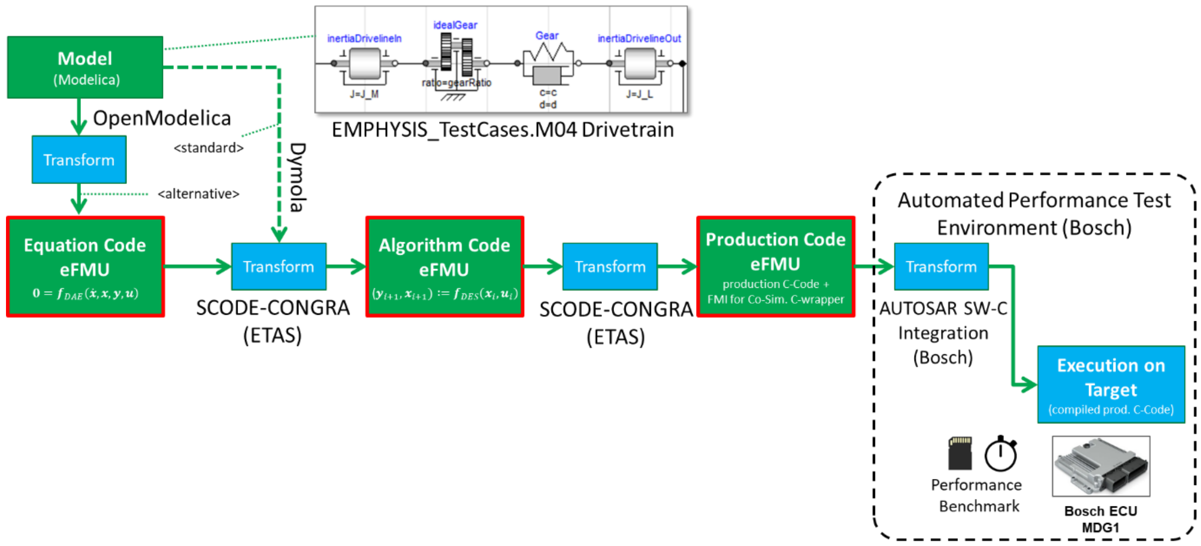


Figure 8: Applied alternative workflow utilizing eFMI for the design of a physics-based controller.

2.3.2. Procedure

The step from an acausal plant model to an *Equation Code* eFMU is demonstrated based on the developed Flat Modelica export of OpenModelica (D4.6). The standardized *Flat Modelica* Language [5] (D3.4) is developed as Modelica Change Proposal (MCP) of the Modelica Association Language Project (MAP_LANG) and is publicly available as branch of the github repository under: https://github.com/modelica/ModelicaSpecification/tree/MCP/0031. The EMPHYSIS partners (Bosch, DS, ETAS, LiU, RISE) have been contributing to this effort.

Starting point of the considered control design use case scenario is the feed-forward controlled open loop system. A common physics-based approach to improve the undesired oscillations of the system is a non-linear feed-forward compensation in combination with a linear PI controller as shown in Figure 9.

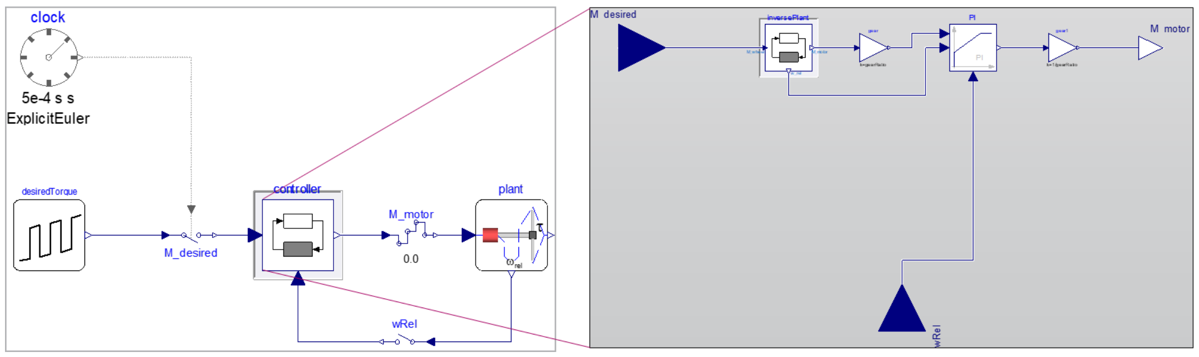


Figure 9: Physics-based approach using a non-linear feed-forward compensation with PI controller

As illustrated in the EMPHYSIS Test Case M04 this type of controller can be designed in a very elegant way by taking advantage of the acausal nature of the Modelica language. As shown in Figure 10, the inversion constraint block from the Modelica Standard Library (MSL) allows to connect the approximated plant model against its original signal flow direction. The load torque at the wheel side is considered as input and the motor torque is considered as output. In other words, the inverse plant model determines the motor torque required to realize the desired wheel torque. Furthermore, the inverse plant computes the relative angular velocity ω_{rel} that is to be expected by the distortion of the driveline. Using ω_{rel} as command input to the PI controller allows to control the actual velocity in a closed loop. The required torque and the output of the PI controller are then added and used as command signal to the motor.

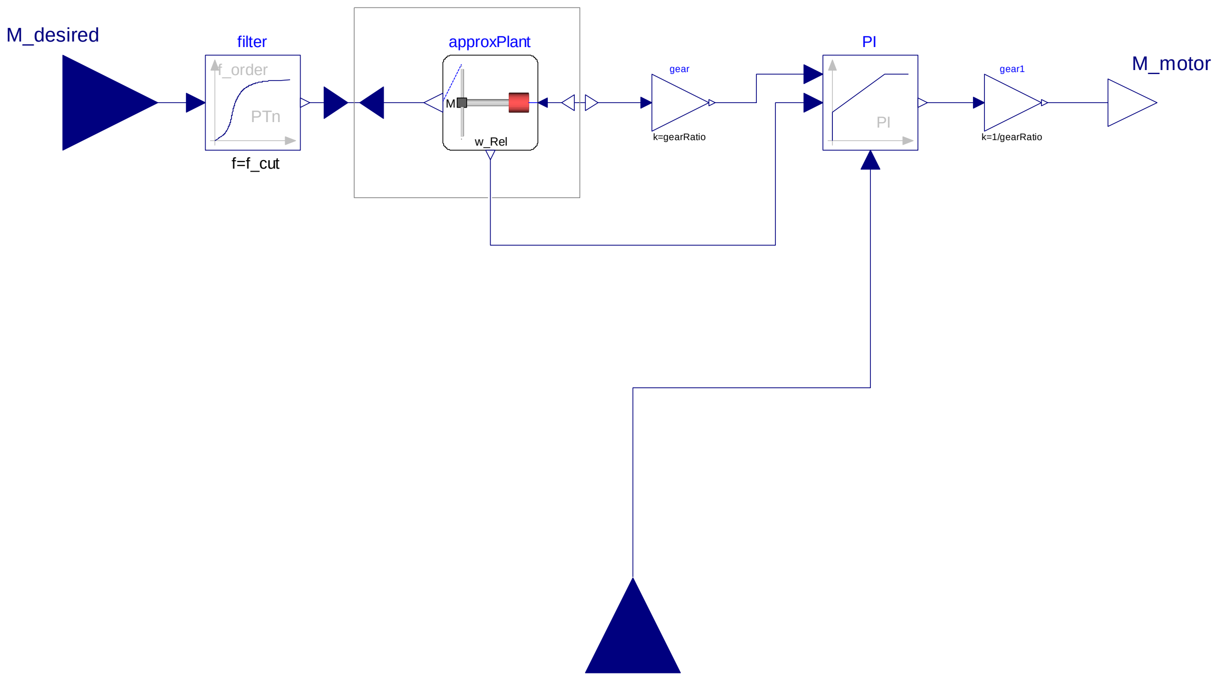


Figure 10: Non-linear controller with inverse plant model in the feed-forward path

Most critical for this architecture is a proper inverse model of the plant, which is able to capture the dominant eigenmodes of the system, has pleasant numerical properties and minimal resource demand such that it can be executed in real-time on an embedded device. This includes a minimal number of states, no non-linear algebraic loops and no numerically stiff differential equations.

Based on the proposed version of the specification, a Flat Modelica export prototype was created in OpenModelica 1.16.29 and used to export the Modelica model from the test cases Library: eFMI_TestCases.M04_DrivetrainTorqueControl.PlantModels.PlantForInversion. The developed Flat Modelica parser was used to convert the model into the native data structure of SCODE-CONGRA in the syq format.

2.3.3. Results

Based on the generated .syq file SCODE-CONGRA successfully generated graphical views for the system and derived so called flows for the further analysis and processing of the equation system.

The equation-based non-Modelica tool SCODE-CONGRA provides modeling and visualization capabilities working directly with the bipartite graph of the equation system as shown in Figure 11. These graph views in addition to the textual view provide full transparency of the structure of the equation system. SCODE-CONGRA also provides features to perform a structural analysis (find over- and underdetermined parts), symbolically derive derivatives and discretize differential equations.

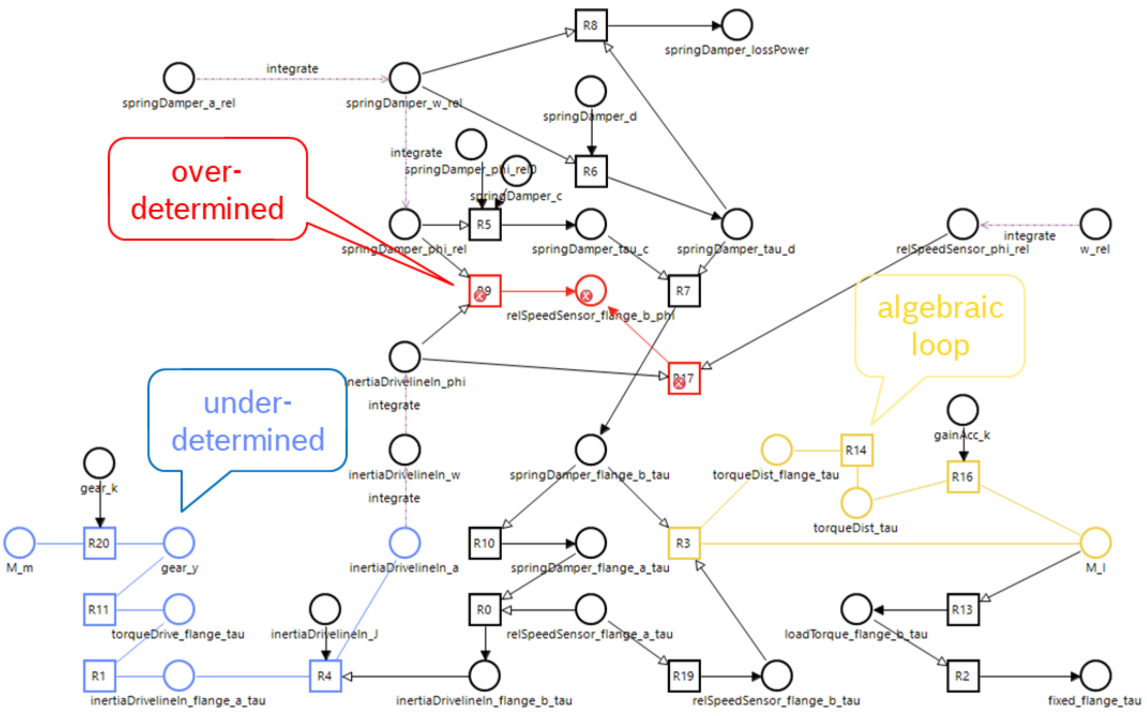
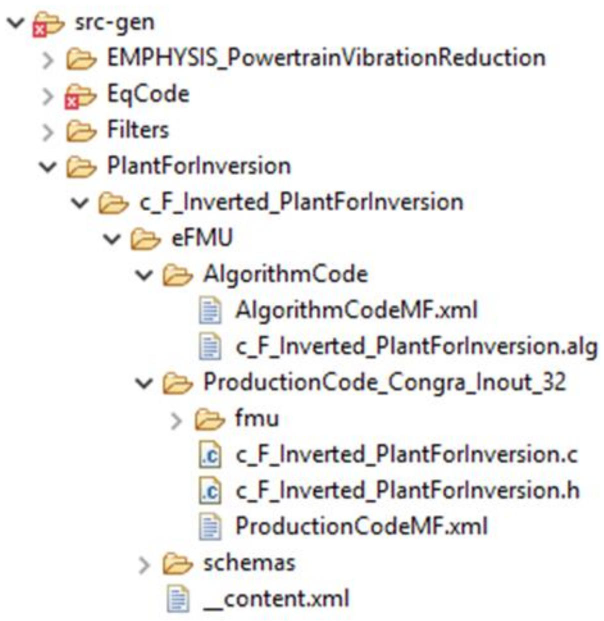


Figure 11: Undirected model of the approximated plant model in SCODE-CONGRA after import of the Flat Modelica code from OpenModelica.

This way a proper computation order has been determined for the imported model considering an inverted signal flow with a modified causality of the input and output signals. The computation graph derived from the flow was ready to be used for code generation.

Code generation in SCODE-CONGRA can target both the generation of GALEC code and embedded C code which can be used for eFMI production code. The production code can optionally be generated as AUTOSAR Software component.



```
src-gen
  EMPHYSIS_PowertrainVibrationReduction
  EqCode
  Filters
  PlantForInversion
    c_F_Inverted_PlantForInversion
      eFMU
        AlgorithmCode
          AlgorithmCodeMF.xml
          c_F_Inverted_PlantForInversion.alg
        ProductionCode_Congra_Inout_32
          fmu
          c_F_Inverted_PlantForInversion.c
          c_F_Inverted_PlantForInversion.h
          ProductionCodeMF.xml
        schemas
        __content.xml
```

Figure 12: eFMU generated by SCODE-CONGRA containing Algorithm Code and Production Code

The generated code can then be integrated with other embedded software, e.g. an AUTOSAR project, to be compiled and deployed on the target.

This completes the whole eFMI workflow (Figure 8), going from a physical model to software on an embedded target.

2.3.4. Summary and Conclusions

In this demonstrator the proposed alternative eFMI workflow has been successfully applied using the tool prototypes of OpenModelica and SCODE-CONGRA. The preliminary Flat Modelica language, proposed for future standardization, has been used as eFMI *Equation Code* language.

The application example illustrated the benefits of the alternative eFMI workflow which enables an assisted transformation of the acausal equation-based model to a computable algorithm. This allows to make informed design decision in the embedded software development process.

The proposed Flat Modelica language has been proven to be a suitable language for the intended eFMI workflow including an Equation Code model representation. The tool chain is applicable also to AUTOSAR software architectures.

2.4. D7.4 Demonstrator for Bosch engine control function

2.4.1. Objective

Model-based diagnosis is a very promising approach for automotive thermal systems to address the need for managing software complexity, reducing risk in an early development stage and increasing the development productivity of delivering robust diagnosis functions for embedded devices (ECU). This demonstrator is illustrating how the proposed Equation Code Model Representation of the new eFMI standard is enabling a new tool chain (see Figure 13) for developing model-based diagnosis functions with more confidence and higher overall efficiency already in a very early stage.

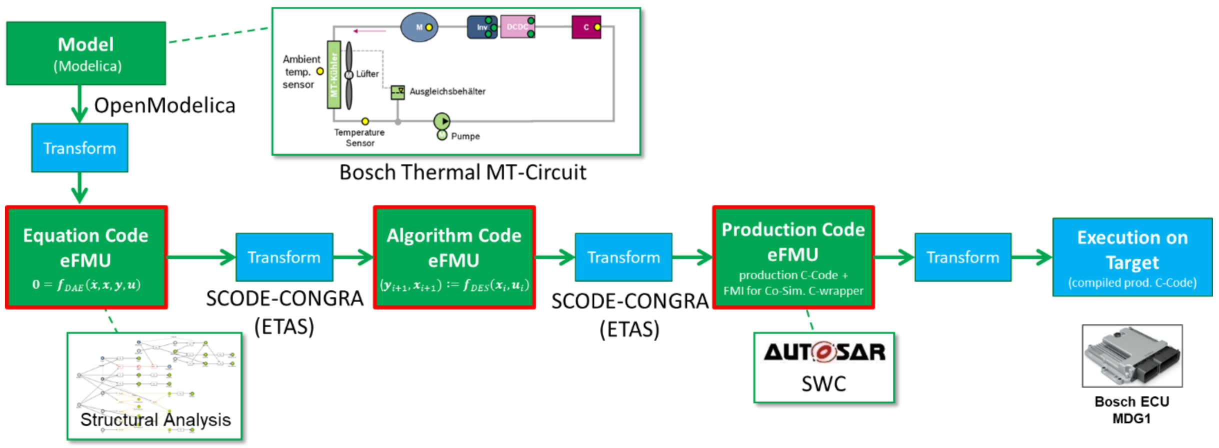


Figure 13: Applied alternative workflow utilizing eFMI for the design of a physics-based controller.

2.4.2. Procedure

Diagnosis functions are aiming to provide indications of the current fault state of the observed system. The fault detection is expected to signal if any of the considered faults is active. Beyond the fault detection does the fault isolation allow to name which of the considered faults is causing the problem. Dependent on the impact of the individual faults on the system and the available sensors it may be that not all faults can be fully isolated but can at least be narrowed down to a set of faults.

The coolant circuit considered in this demonstrator is from a Bosch application project of a hybrid electrical vehicle with range extender. As shown in Figure 14, the thermal system consists of two circuits (MT and LT) that can be operated in different modes. In the split mode the two circuits are independent from each other. The MT circuit considered in this demonstrator is cooling the motor (EM) and generator (Gen) together with their corresponding inverters (Inv, GenInv) in a serial fashion. The heat is transferred to the air flow through the radiator. The air flow depends on the vehicle speed and the speed of the radiator fan.

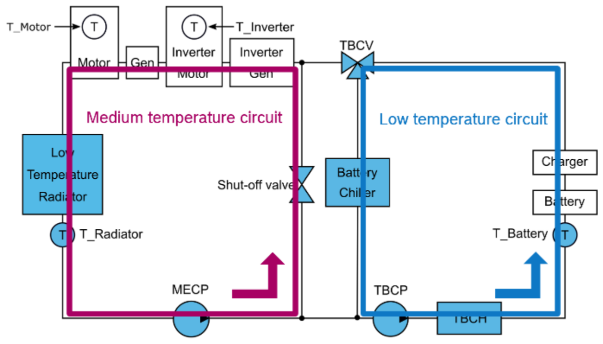
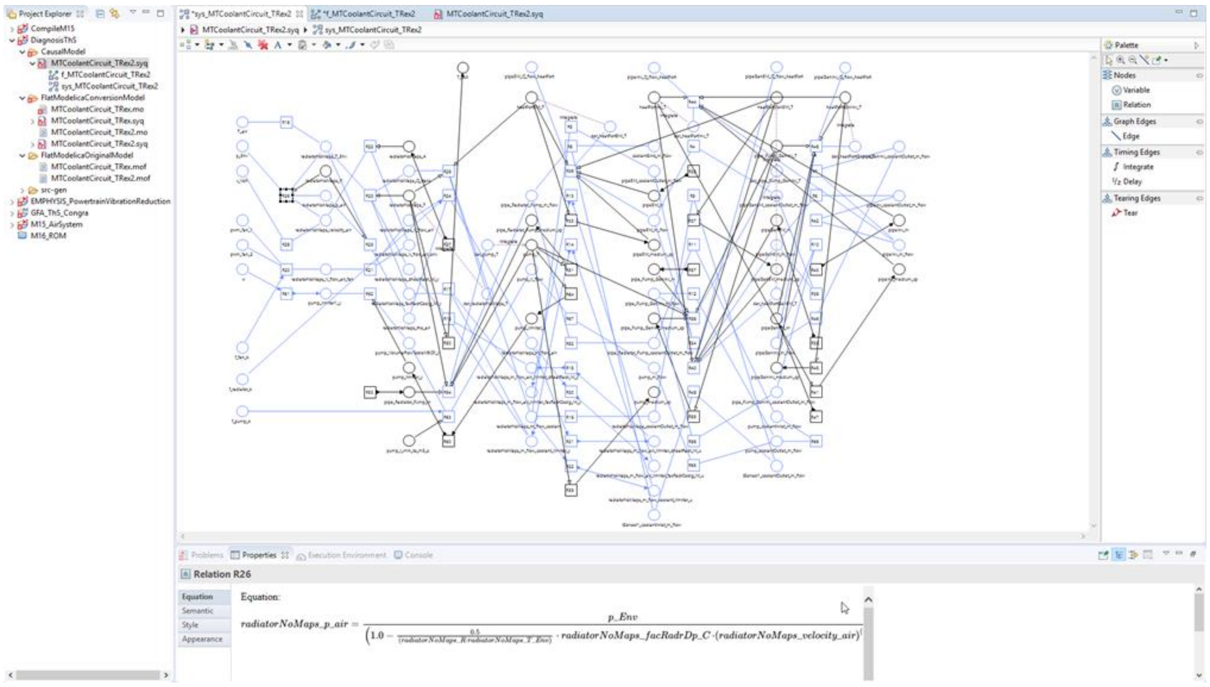


Figure 14: Thermal system of the range extender EV in split mode with the medium temperature (MT) and low temperature (LT) circuits being disconnected from each other by the shut-off valve and the switching valve.

Based on the newly developed IdealCoolantFlow Library the MT circuit model has been built up. The circuit model is stimulated by measured control signals from a road test and simulated to validate the system responses against the measured sensor signals.

Following the proposed workflow shown in Figure 13 the coolant circuit model has been taken as starting point of the further procedure. In the same way as described in the demonstrator D7.3 in the previous section, the equations have been extracted using the proposed *Equation Code* language "Flat Modelica" [5] (D3.4) exported by the OpenModelica prototype (D4.6). The *Equation Code* import prototype of SCODE-CONGRA (D5.3) is then used to convert the imported equations into its internal syq format and generate the corresponding dependency graph as shown in Figure 15.



$$radiatorNoMaps\_p\_air = \frac{p\_Env}{\left(1.0 - \frac{0.5}{(radiatorNoMaps\_R \cdot radiatorNoMaps\_T\_Env)} \cdot radiatorNoMaps\_facRadrDp\_C \cdot (radiatorNoMaps\_velocity\_air)\right.}$$

Figure 15: Dependency graph in SCODE-CONGRA (ETAS) derived from the imported and converted Flat Modelica coolant circuit model.

In addition to the above described signal causality, SCODE-CONGRA provides so-called "Diagnostic" features. These allow to apply a diagnosis-specific semantics to variables and relations. In addition to control inputs and available sensor signals, that is the "diagnostic inputs", the system is enhanced by a declaration of input variables that allow to trigger "faults". Based on this system definition, a "diagnostic flow" as shown in Figure 16 can be derived.

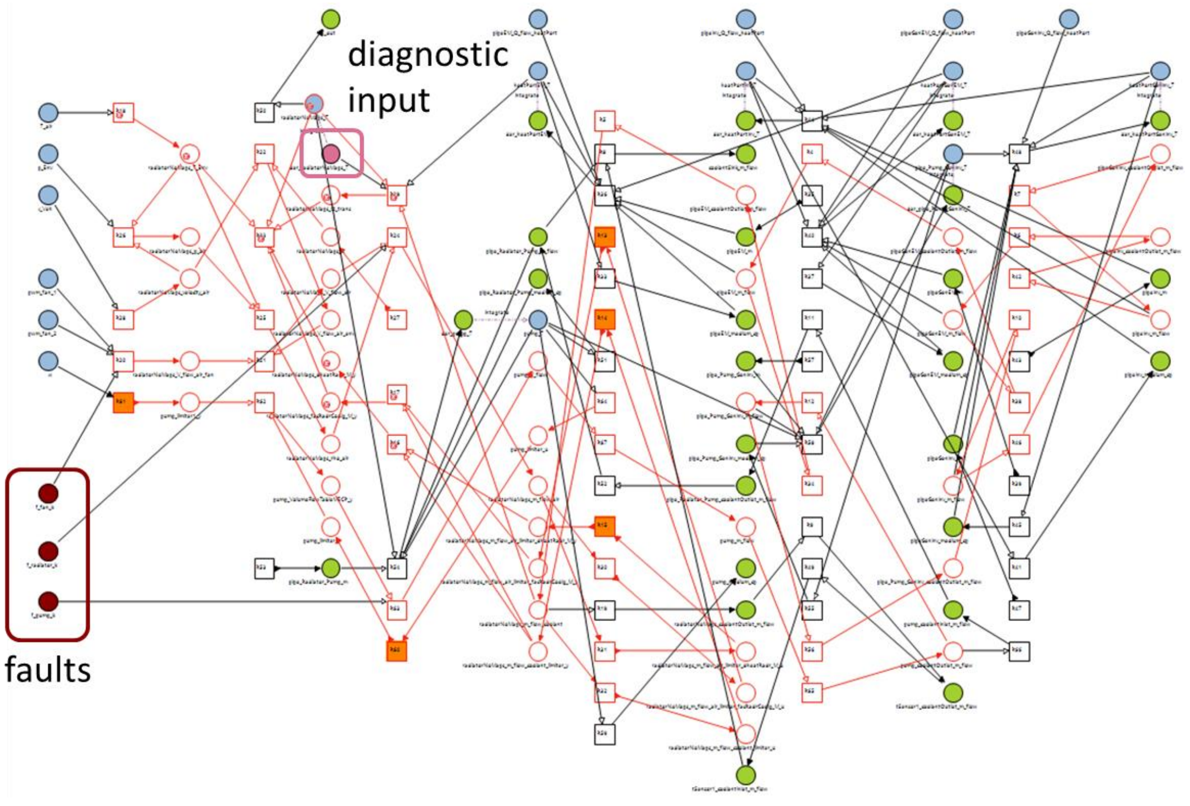


Figure 16: Diagnostic flow considering the gradient of the coolant temperature as diagnostic input.

2.4.3. Results

The imported and enhanced model allowed to run a complete structural analysis resulting in the full incidence matrix automatically generated after all temperature gradients of the temperature sensor are marked as diagnostic input and the resulting overdetermined subset have been resolved by introducing five additional residual equations to balance the system. This incidence matrix allows to find out which faults can be isolated and which residuals are required to detect and isolate them. After an appropriate selection has been applied based on this analysis the corresponding computational graph is automatically compiled. This graph is then used for the generation of production code and its conversion into an eFMU together with the corresponding code files and manifest files (see Figure 17).

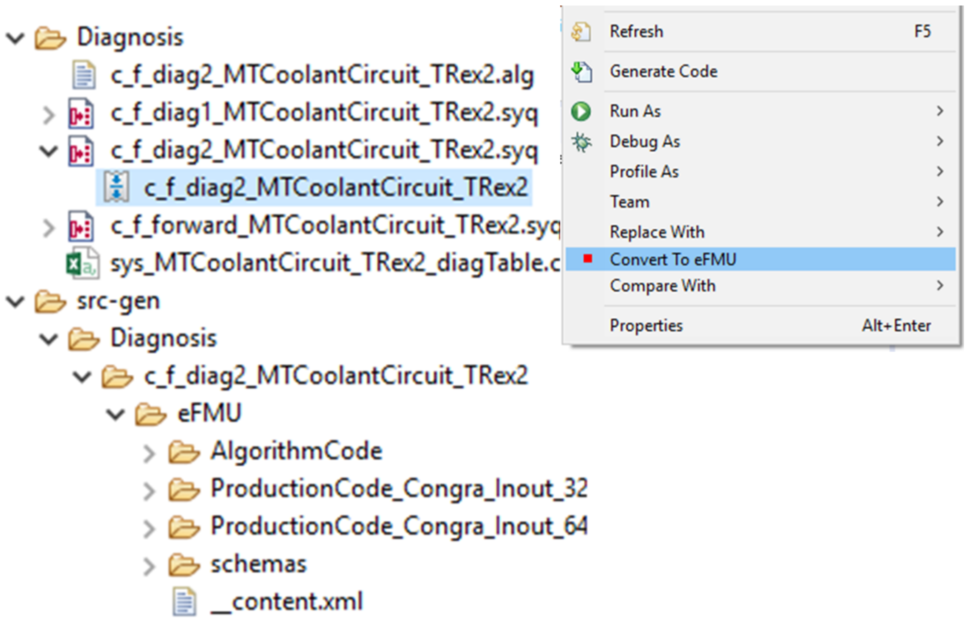

Diagnosis
c_f_diag2_MTCoolantCircuit_TRex2.alg
c_f_diag1_MTCoolantCircuit_TRex2.syq
c_f_diag2_MTCoolantCircuit_TRex2.syq
c_f_diag2_MTCoolantCircuit_TRex2
c_f_forward_MTCoolantCircuit_TRex2.syq
sys_MTCoolantCircuit_TRex2_diagTable.c
src-gen
Diagnosis
c_f_diag2_MTCoolantCircuit_TRex2
eFMU
AlgorithmCode
ProductionCode_Congra_Inout_32
ProductionCode_Congra_Inout_64
schemas
__content.xml
Refresh F5
Generate Code
Run As
Debug As
Profile As
Team
Replace With
Convert To eFMU
Compare With
Properties Alt+Enter

Figure 17: Computation graph generated from a diagnostic flow corresponding to the desired residual and generated production code according to the eFMI specification.

2.4.4. Summary and Conclusions

This demonstrator is presenting the IdealCoolantFlow Modelica library for thermal systems designed for the component-oriented modeling of coolant circuits of HEVs and EVs capable for real-time simulations. The library is used to create and validate the behaviour of the MT coolant circuit of a HEV (range extender). The parameters of the model are fitted to measurement data.

The equations of the MT coolant circuit are derived from the model using the proposed Flat Modelica language as eFMI *Equation Code* by OpenModelica. This code is parsed and converted by SCODE-CONGRA. A structural analysis has been successfully applied and revealed that all three faults can be detected but only one fault can be isolated.

The found residuals have been successfully exported as *Algorithm Code* and *Production Code* eFMU.

The current Flat Modelica prototypes do not yet properly support Modelica tables and records so that only a simplified variant of the coolant circuit model has been processed. Therefore, the quantitative study of the residual responses could not yet be performed to validate the diagnosis function.

The demonstrator was able to prove that the proposed eFMI workflow is able to perform sophisticated structural analysis in an early project state in a highly efficient way by leveraging the benefits of a component-oriented modelling approach using the developed IdealCoolantFlow Modelica library.

2.5. D7.5 Demonstrator for gasoline engine TDC air filling estimation

The demonstrator shows the development of a virtual sensor of air filling the cylinder of a 4 strokes internal combustion engine with variable valve timing.

In this demonstrator an eFMI is used as container of a Neural Network (NN) that provides a parameter in a nonlinear prediction model, which is then used in an Extended Kalman Filter (EKF). The EKF is needed to correct the error between the available measured air mass right after the air filter and the actual air mass filling the cylinder. The EKF is implemented as virtual sensor on real hardware and tested on a prototype vehicle.

This work was conducted in close collaboration with Renault SAS, Siemens, OSE and FH electronics. Renault provided the use case and the engine data. OSE developed the physical plant model. Siemens developed the NN model and did the export of the NN model as eFMU. Renault was responsible for the design of the EKF and FH electronics for its implementation in the real hardware and prototype vehicle.

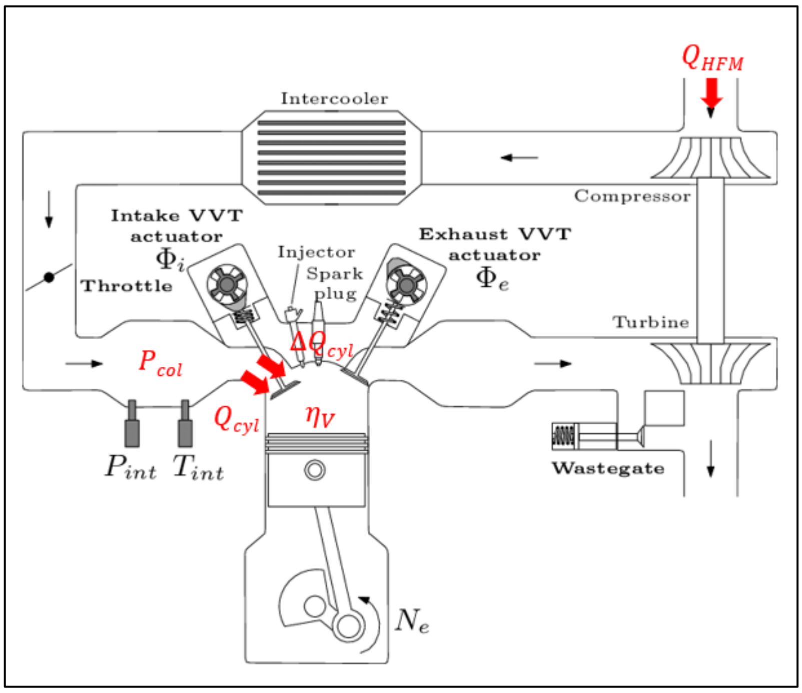


Figure 18: Schematic view of a gasoline Engine

The model is centered on the intake pressure P_{col}. The air filling factor η_V is provided by an external model. In a desk top study η_V can be computed by the AMESim model. For implementation in the vehicle it must be replaced by a surrogate model, in this demonstrator the neural network. Other variables used in the model are the engine speed N and the actual mass flow entering the cylinder Q_{cyl}. The measured air mass flow Q_{HFM} approximates Q_{cyl}.

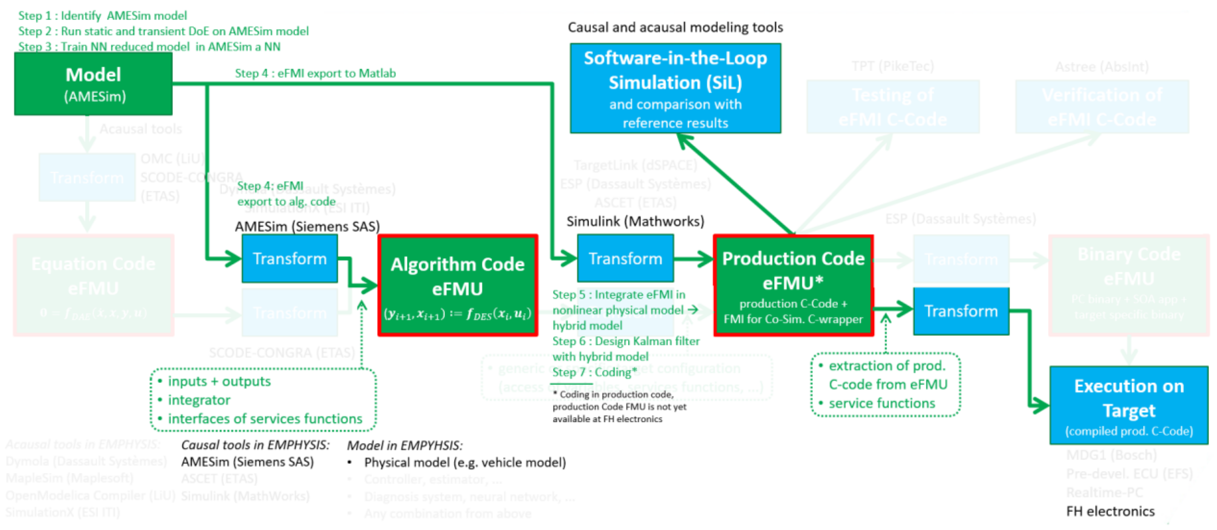

Step 1 : Identify AMESim model
Step 2 : Run static and transient DoE on AMESim model
Step 3 : Train NN reduced model in AMESim a NN
Model (AMESim)
Step 4 : eFMI export to Matlab
Causal and acausal modeling tools
Software-in-the-Loop Simulation (SiL) and comparison with reference results
Step 4: eFMI export to alg. code
AMESim (Siemens SAS)
Transform
Algorithm Code eFMU
$(y_{i+1}, x_{i+1}) := f_{DES}(x_i, u_i)$
Simulink (Mathworks)
Transform
Production Code eFMU* production C-Code + FMI for Co-Sim. C-wrapper
Step 5 : Integrate eFMI in nonlinear physical model → hybrid model
Step 6 : Design Kalman filter with hybrid model
Step 7 : Coding*
* Coding in production code, production Code FMU is not yet available at FH electronics
Transform
• extraction of prod. C-code from eFMU
• service functions
• inputs + outputs
• integrator
• interfaces of services functions
Execution on Target (compiled prod. C-Code)
FH electronics
Causal tools in EMPHYSIS:
AMESim (Siemens SAS)
Model in EMPYHSIS:
• Physical model (e.g. vehicle model)

Figure 19: eFMI Workflow applied to the TDC air filling estimator demonstrator

Step 1: AMESim Model: Calibration of one-degree crankshaft HF model using test cycles of air filling, combustion, thermal losses, etc. The model is validated on Variable Valve Time sweeps and transient driving cycles (NEDC, WLTC, RDE).

Step 2: DoE with calibrated AMESim model using static or dynamic tests, e.g. RDE type running cycles. In the following step the model is used as a virtual engine bench to build a database to train the NN based scale model. Being indestructible, it allows exploring larger areas than a real engine and enables massive testing.

Step 3: Build of the NN filling model with native AMESim tool. The tool performs dedicated DoE to train NN. Correlation between data from Step 1 AMESim HF model and reduced NN model shows the absolute errors at the expected precision.

Step 4: eFMI export with the AMESim tool. The eFMU NN scale model is now ready to be integrated in a MATLAB/Simulink code.

Step 5: Global model: Integration of eFMI NN in the nonlinear physical global model → hybrid model.

Step 6: EKF integration using the Step 5 prediction model and measurements to better accuracy

- The predictor is open loop, based on test data generated from the AMESim model.
- To absorb remaining dispersion, loop back to a reliable measurement on the engine, here, the air flow before air filter.
- Use of a Kalman filter to have a certain level of optimality of observation.
- As the air filling problem is slightly nonlinear, usage of an EKF

Step 7: Integration on the vehicle

2.6. D7.6 Demonstrator for high frequency throttle position estimation

The demonstrator shows the development of a virtual sensor of the throttle position of a 4 strokes internal combustion engine.

The eFMI is used as container of a Neural Network (NN) that can then be implemented on real hardware and tested on a prototype vehicle.

This work was conducted in close collaboration with Renault SAS, Siemens, and FH electronics. Renault provided the use case and the data. Siemens developed the physical plant model, the NN model and did the export of the NN model as eFMU. FH electronics did the implementation in the real hardware and prototype vehicle.

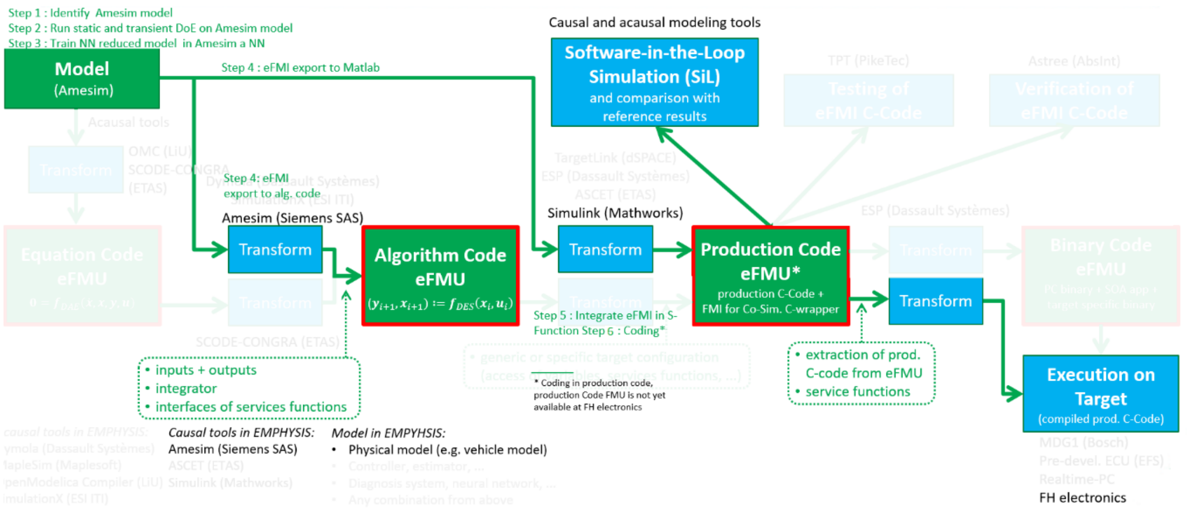

Step 1 : Identify Amesim model
Step 2 : Run static and transient DoE on Amesim model
Step 3 : Train NN reduced model in Amesim a NN
Model (Amesim)
Step 4 : eFMI export to Matlab
Step 4: eFMI export to alg. code
Amesim (Siemens SAS)
Transform
Algorithm Code eFMU
$(y_{i+1}, x_{i+1}) := f_{DES}(x_i, u_i)$
• inputs + outputs
• integrator
• interfaces of services functions
Causal and acausal modeling tools
Software-in-the-Loop Simulation (SiL) and comparison with reference results
Simulink (Mathworks)
Transform
Production Code eFMU* production C-Code + FMI for Co-Sim. C-wrapper
Step 5 : Integrate eFMI in S-Function Step 6 : Coding*
* Coding in production code, production Code FMU is not yet available at FH electronics
• extraction of prod. C-code from eFMU
• service functions
Transform
Execution on Target (compiled prod. C-Code)
FH electronics
Causal tools in EMPHYSIS:
Amesim (Siemens SAS)
Model in EMPYHSIS:
• Physical model (e.g. vehicle model)

Figure 20: eFMI Workflow applied to the throttle position estimator demonstrator

Step 1: Collect a set of data from the physical system.

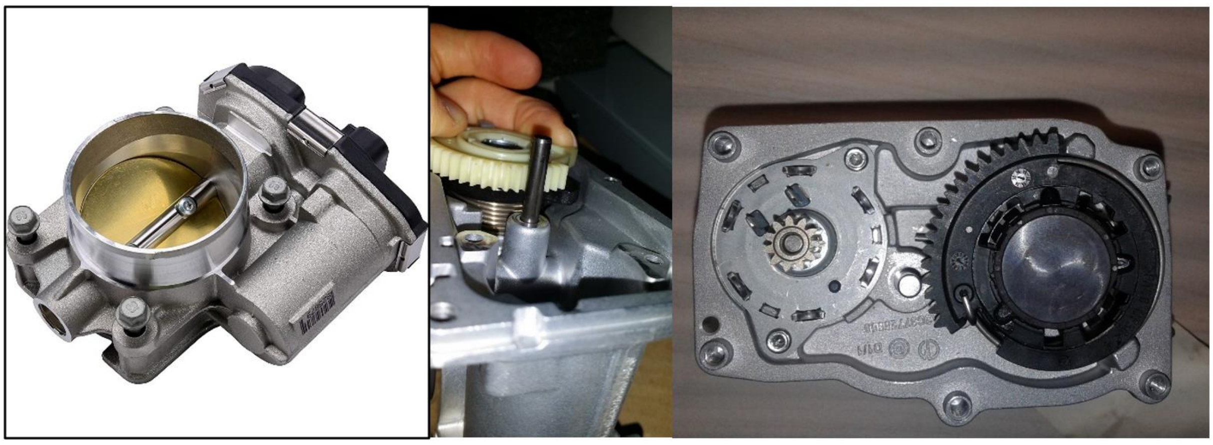


Figure 21: On desk test rig of throttle

Step 2: Build the physical plant model and identify all model parameters.

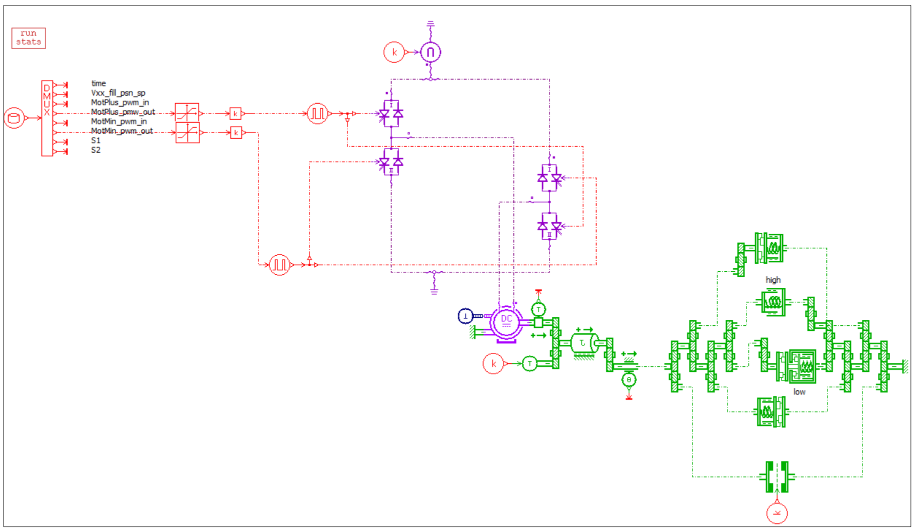
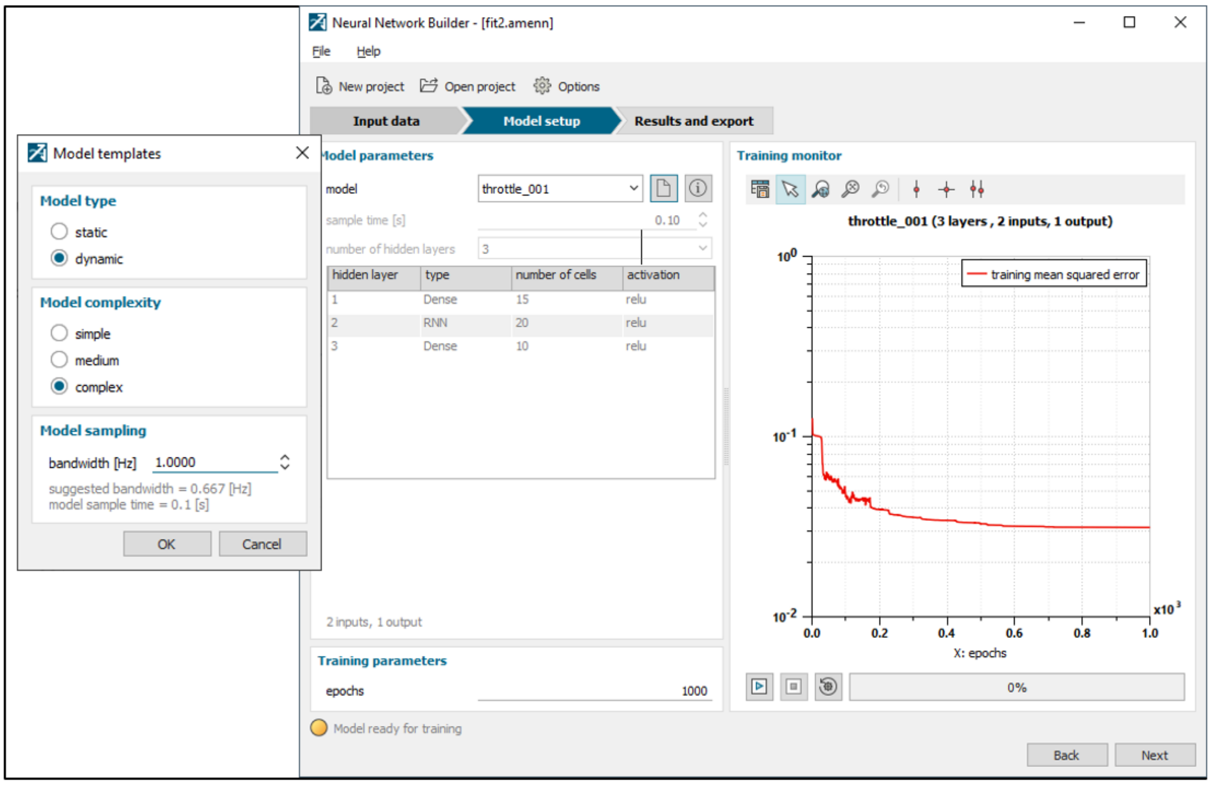


Figure 22: AMESim High Frequency throttle model

Step 3: Build a neural network (NN) scale model on the physical plant model.



Model templates

Model type

static

[✓ dynamic]

Model complexity

simple

medium

[✓ complex]

Model sampling

bandwidth [Hz] 1.0000

suggested bandwidth = 0.667 [Hz]
model sample time = 0.1 [s]

Model parameters

model	throttle_001
sample time [s]	0.10
number of hidden layers	3

hidden layer	type	number of cells	activation
1	Dense	15	relu
2	RNN	20	relu
3	Dense	10	relu

2 inputs, 1 output

Training parameters

epochs 1000

Model ready for training

Training monitor

throttle_001 (3 layers , 2 inputs, 1 output)

Figure 23: AMESim interface to define architecture of NN model and to mentor training

Step 1 is executed on test rig whereas the Steps 2 and 3 are done using a personal computer running the commercial tool AMESim.

Step 4: Export the NN model from AMESim to the commercial tool Matlab/Simulink. The eFMU is used as a container that is then used in a Simulink S-function. The S-function is directly used as virtual sensor.

Step 5: Implementation of the S-function in the commercial engine control unit of FH electronics.

The demonstrator shows the usage of the virtual sensor running in the FH electronics control unit operating in a real car in real driving conditions.

2.7. D7.7 Demonstrator for vehicle dynamics control

The goal of this demonstrator is to validate the full eFMU toolchain by using a neural network model coupled to a controller for semi-active damper regulation.

The physical model used for this work is a neural network (NN) model of the system which is trained using the empirical data obtained through several road profiles and damper force excitations. The semi-active suspension system is regulated using the parametrized Nonlinear Model Predictive Control (pNMPC) controller which is coupled with the NN model. Under the scope of this project, an OpenSource toolbox was developed to automatically generate the C code/Simulink S-function of the controller. The final eFMU code was validated via simulations on Simulink and on RT embedded target platforms such as dSPACE MicroAutoBox II and the INOVE testbench at GIPSA-lab.

This work was conducted in close collaboration with GIPSA-lab, Siemens and dSPACE. Siemens developed the NN model of the dynamics of the vehicle. Siemens and dSPACE supported the conversion of the NN model into eFMU *Algorithm Code* and *Production Code*. GIPSA-lab was responsible for the control algorithm development and the full-scale integration on the embedded target systems. All the methods and tools are compliant with the eFMU module and standard C code.

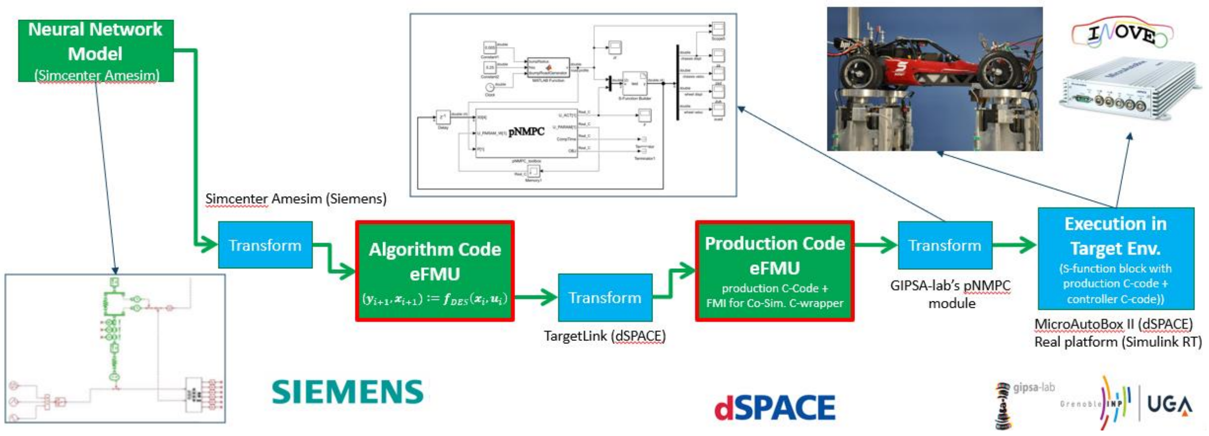


Figure 24: eFMI Workflow applied to vehicle dynamics control demonstrator.

- Step 1: Development of an AMESim model, representing the dynamics between chassis, dampers, wheels and road.
- Step 2: Transformation of the AMESim model into an NN model with the Neural Network Builder from AMESim.

- Step 3: Transformation of the NN model into *Algorithm Code* eFMU with AMESim.
- Step 4: Transformation of the *Algorithm Code* eFMU into *Production Code* eFMU by using TargetLink from dSPACE.
- Step 5: Integration of the eFMU in an S-function for Simulink with the S-function builder. Integration of the eFMU with the controller with the pNMPC toolbox.
- Step 6: Deployment into the MicroAutoBox II from dSPACE, by using ControlDesk.

2.8. D7.8 Demonstrator for dual clutch transmission

This demonstrator shows how to use eFMI in the development of a virtual sensor, here, a virtual sensor of the dynamical properties of a transmission with two coupled clutches, contributed by Daimler AG, a member of the EMPHYSIS Advisory Board, in collaboration with Dassault Systemes, dSPACE and Maplesoft.

In this demonstrator the eFMU is used as a container for a simplified plant model. This simplified model is used as part of an observer, which uses the (previous) control signals and the sensor signals from the plant, as well as historical state estimates, to estimate the current state of the clutches (speed, acceleration, and torque). These state estimates will be used by a controller to generate the new control signals.

The biggest contribution of the dual clutch use case is to draw the project’s attention to a physical model which currently is in focus in real applications in the automotive industry.

The friction clutches in the transmission models demand very specific and highly complex numerical solutions. A friction clutch is modeled as a very stiff and discontinuous equation system and a successful solution to this numerical problem will enable a much broader applicability of the eFMI standard to other industrial applications.

Daimler has performed a successful integration of the generated GALEC code of the dual clutch system on a dSPACE HiL system. Dassault Systèmes has done the development of the required numerical methods and the simulations used for comparison of the results in Dymola.

Applying eFMI is only one of many design steps of the virtual sensor design procedure that has been used to build the demonstrator. The following prerequisite steps have been conducted before eFMI was used in the design procedure:

1. Collect a set of data from an actual transmission. This step is executed on an engine test bench. The data covers all relevant operating conditions in terms of range and dynamics.
2. Build a physical high-fidelity vehicle model and identify all model parameters. Validate the resulting model against the actual engine and transmission.
3. Build a simplified plant model for the transmission to be used in the observer. Identify its model parameters from the high-fidelity vehicle model.

Step 1 is executed on a test bench whereas Steps 2 and 3 are done using a personal computer running the Dymola (Dassault Systèmes) simulation software. After collecting all prerequisite data and modelling the physics system, eFMI tooling has been used as follows:

4. Export the simplified plant model from Dymola as an *Algorithm Code* eFMU.

5. From the *Algorithm Code* eFMU, a *Production Code* eFMU is generated, using TargetLink. Also, an FMU wrapper that can be used for SiL simulation in Dymola is generated.
6. Deployment of the eFMU/FMU on a target system (dSpace Scalexio) for HiL simulation using ConfigurationDesk (dSpace) and PROVEtech (AKKA Technologies).

The following steps are still future work:

7. The eFMU/FMU is to be used inside a virtual sensor, which in turn is used inside a controller.
8. Deployment of the eFMU/FMU on the final target prototype TCU (Schaeffler) using a Matlab/Simulink (MathWorks) based environment.

Successful simulation of the simplified dual clutch model as a clocked model has been achieved. This was not possible when the model was first provided, and it is a pre-requirement for successful eFMI export.

The simulation still exhibits unacceptable artefacts, however. Several additional requirements for eFMI, such as (differentiable) lookup tables and stiff solvers, have been identified over the course of the project. Support for these was partially implemented in Dymola, but further work is required. The model imposes a highly stiff system that is not suited for embedded code generation; further work on the modeling side and on the supporting tools (e.g., numerical integrators) is required.

Parts of the eFMI tool chain – from (simplified) plant model to GALEC Code to *Production Code* – have been validated for this use case, but no comparative simulation of the generated *Production Code* eFMU with the high-fidelity vehicle model have been performed, due to the above simulation artefacts.

The integration was carried out using a C-Code FMU as a container and the eFMU was integrated into an existing model that is currently in use for software testing. The successful integration and loading on the HiL simulator show the ability to deploy code that was created with the eFMI toolchain to a real-time target.

The project developers recognized that the longer-term goal of supporting eFMU export for the dual clutch model is a hard problem beyond the scope of the EMPHYSIS project. Daimler understands this and considers the progress made by project developers towards this goal within the project, as summarized above, a success.

2.9. D7.10 Demonstrator for Hybrid Air Virtual Sensor

A virtual sensor for the estimation of the torque of an engine in a hybrid drive train has been created in the form of a neural network and tested.

During the demonstrator the transformations as shown in Figure 25 were followed.

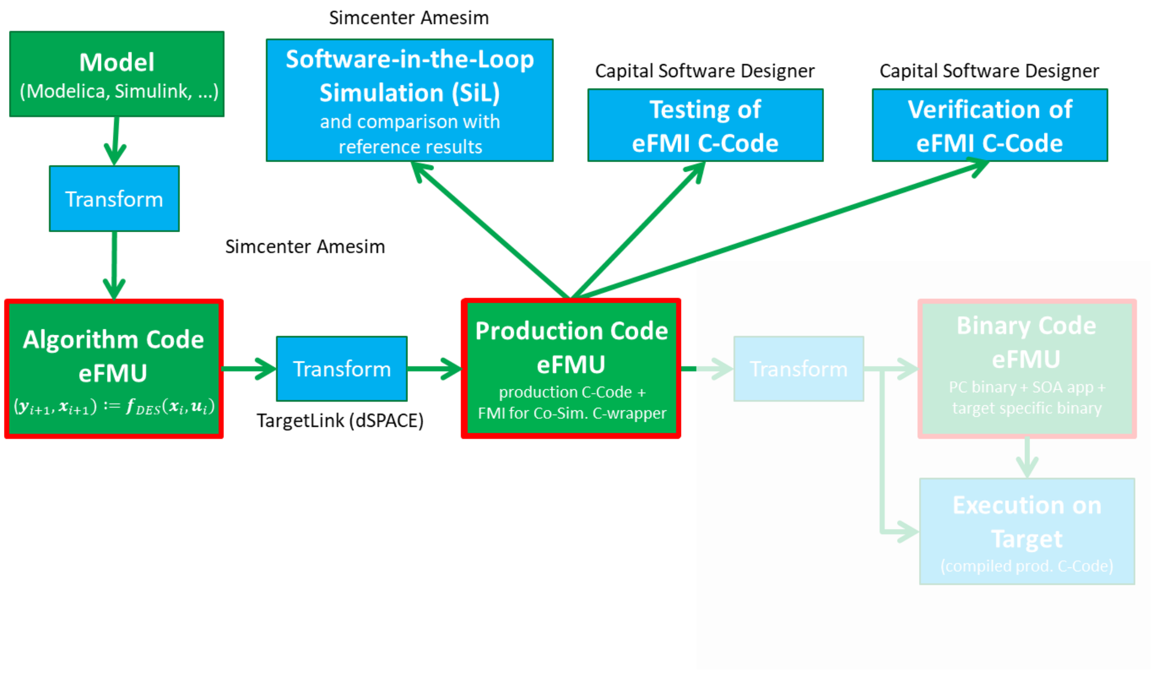

Simcenter Amesim
Model
(Modelica, Simulink, ...)
Software-in-the-Loop Simulation (SiL)
and comparison with reference results
Capital Software Designer
Testing of eFMI C-Code
Capital Software Designer
Verification of eFMI C-Code
Transform
Simcenter Amesim
Algorithm Code eFMU
$(y_{i+1}, x_{i+1}) := f_{DES}(x_i, u_i)$
Transform
TargetLink (dSPACE)
Production Code eFMU
production C-Code + FMI for Co-Sim. C-wrapper
Transform
Binary Code eFMU
PC binary + SOA app + target specific binary
Execution on Target
(compiled prod. C-Code)

Figure 25: e-FMI path for D7.10 Demonstrator

Starting from an Amesim model it was transformed to *Algorithm Code* through a neural network. From there dSPACE was used to transform the *Algorithm Code* into *Production Code*. Afterwards, it was imported into CSD where the production code was tested and verified.

During this demonstrator particular attention has gone into translating simulation models into neural networks. This led to some interesting conclusions on assessing and comparing the trainings data upfront to understand its qualitative effect on the trained network. It allowed as well to start generating the trainings data in such a way that it would improve the performance of the network. This led to the root mean square (RMS) error of the new network to be only half as big and reduce the max RMS error by more than 25% on the full range of operation.

Furthermore, attention has been spent on analyzing the design process, identify each critical step in the process and define methodologies and operations that would help the user working with this technology. To this end eFMI has been integrated with the Validity frame concept of the University of Antwerp.

2.10. D7.12 Demonstrator for semi-active damping controller with hardware tests

A vertical dynamics controller for semi-active dampers developed by the DLR Institute of System Dynamics and Control (DLR-SR) serves as example for illustrating and testing the eFMI tool chain. The demonstrator covers several aspects of model-based nonlinear control. The considered semi-active dampers in a real vehicle are controlled by current setpoints every 1 ms. The controller consists of two parts:

- a nonlinear Kalman filter to estimate non-measured variables and
- a nonlinear controller based on a dynamic nonlinear inverse model.

The demonstrator (see Figure 26) covers the entire eFMI tool chain. The tool chain starts with Modelica models of the vehicle, the controller and the prediction model of the Kalman filter and results in production C code, which is compiled and executed in real time on a small series electronic control unit (ECU) in vehicle driving tests.

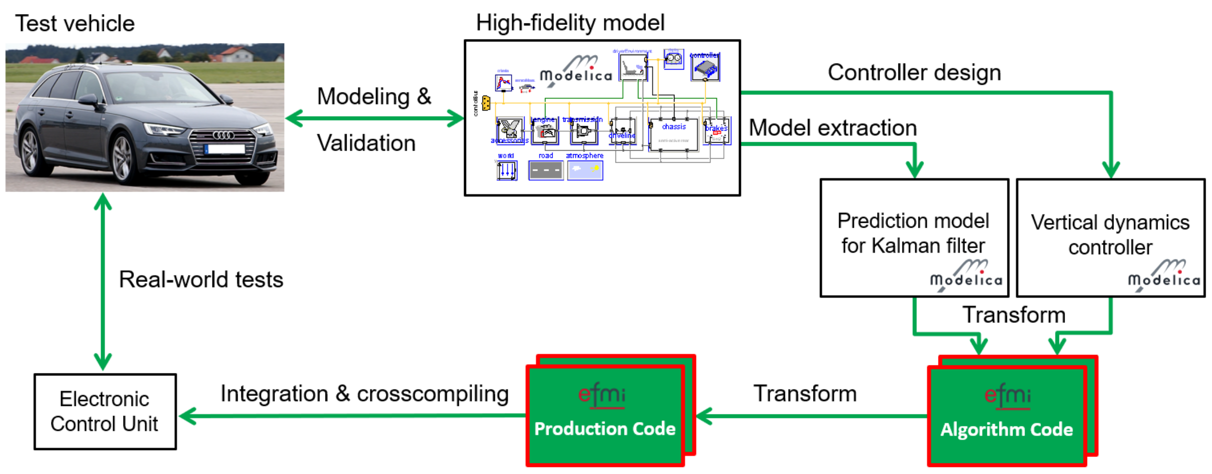


Figure 26: Process of designing, generating and testing a semi-active damping controller using eFMI

In the early design stage, the controller is tested and parametrized in a closed loop simulation utilizing a high-fidelity vehicle model. This high-fidelity model comprises several physical domains such as multi-body mechanics of the chassis or a thermal brake model, all implemented using Modelica. The vehicle model is parametrized using data provided by EFS and cross-checked with an existing table-based vehicle dynamics model.

Both the controller model and the prediction model for the nonlinear Kalman filter are exported to eFMI Algorithm code by a Dymola prototype (Dassault Systemes). Due to special requirements of the applied Kalman filter algorithms, some manual modifications to the *Algorithm Code* of the prediction model are necessary. In the next step, the eFMI *Production Code* is generated by a TargetLink prototype (dSPACE). The C code of the prediction model is interfaced to a generic nonlinear Kalman filter C library developed by DLR-SR. All C-code parts (controller, prediction model and Kalman filter library) are integrated into the complete software framework of the ECU to control the semi-active dampers of the vehicle.

Open-loop tests with the generated C code as well as open-loop tests with the integrated code on the ECU show the expected behavior of the functions. To validate the controller as well as the nonlinear Kalman filter in real operation, an Audi A4 is equipped with additional sensors and a rapid-prototyping-platform (dSPACE MABX II) for sensor data processing. Different maneuvers with driver-side excitations and with additional road-side excitations have been carried out to separately evaluate the diverse functions of the controller. For the validation of the controller in real vehicle operation, an identical trajectory has been driven at the same speed with an active and inactive controller (passive system). A fixed damper characteristic curve is chosen for the passive system, that is the damper current is set to a constant value. In the results the performance of the controller is clearly visible by damped roll and pitch angles of the vehicle and a reduced acceleration at the center of gravity.

The demonstrator shows the successful application of a tool chain starting with a controller or prediction model in Modelica up to compiled, tested and integrated production code on a small series ECU for semi-active damping including driving tests on a vehicle.

2.11. D7.13 Demonstrator for virtual sensor for electric machine control

This demonstrator from Volvo Cars shows how to use eFMI in the development of a virtual sensor of the transmission in an electric vehicle. The virtual sensor is used to improve the control software of the electric machine in order to increase the drivability and durability of the vehicle.

One issue that is the focus of this demonstrator is the mitigation of backlash and resonance frequencies in the electric driveline. This can be caused by incomplete information about the state of the complete driveline, including driveshaft and wheel, when requesting torque from the electric machine. In addition to making the driving experience uncomfortable, the occurrence of e.g. backlash can cause decreased durability of mechanical components.

To solve the problem, more physical sensors could be used to increase the knowledge of the state of the vehicle. However, adding sensors is a challenge due to, among other things, lack of space, cost, and high latency in the communication network. An alternative is to use an advanced physics-based model as a virtual sensor that can be used by the electric machine control software to increase the amount of information available. Here, the eFMI standard adds the important advantage of being able to reuse existing models of the physical hardware in question, in this case the transmission. The reuse of models that are already verified and validated against measurements is a great advantage when it comes to reliability and resources in terms of time.

The demonstrator is set up to add a virtual sensor to the controller of an electrical machine. The control of delivering torque through the driveline needs to have a signal back from the wheels to better judge possible vibrations and other behaviors that are unwanted. Vibrations lead to increased wear on components of the transmission and feel and experience of the drive for the people in the vehicle that are unpleasant. By adding a virtual sensor that measures signals on a physical model of the transmission, one can get improved performance compared to making decisions based on a coarser signal coming from the ECU that measures the movement of the wheels, removing the update frequency coming from the configuration of the communication network and any latencies from the ECU software and possible gateways. By sampling the virtual sensor, one gets a better view of the state of the transmission, with less wear on the transmission and better performance.

In the demonstrator, the eFMI toolchain was successfully used to generate production code of a physical model of the transmission of a battery electric vehicle (BEV) to be used in the control software for the electric machine. The original acausal, equation-based physics model was transformed to a causal, discretized form (eFMI GALEC code / *Algorithm Code* container) using the Dymola Modelica IDE and then transformed to *Production Code* using three different eFMI *Production Code* generators: ESP, TargetLink, and SCODA-CONGRA. To verify that the generated production code representation of the physical system behaves in a similar way to the original model, a Software-in-the-Loop (SiL) simulation was conducted with the *Production Code* generated by ESP. The results from the SiL simulation was compared to the offline results from the original model to ensure consistent behavior. Additionally, all three production codes have been tested using TPT, based on an eFMI *Behavioral Model* container derived from the original Modelica test scenarios.

The objective of evaluating the eFMI toolchain for an industrial use case within the automotive industry was successful and shows promise when it comes to reducing resources and errors when using physical models as virtual sensors or other parts of software control algorithms. The main advantages from Volvo Cars' perspective is that an existing offline model of the physical system in question can be used. The reuse of existing physics models decreases the development time and reduces the risk of error, since only one model has to be developed, maintained, and validated. To that end, it is important that existing physics models can be transformed to real-time suited solutions. The automatic handling of mixed systems of equations, as demonstrated by Dymola's GALEC code generator, is impressive.

Another major advantage from Volvo Cars' perspective is that the eFMI standard allows for different tools to be part of the tool-chain. The ability to use tools from different vendors in the tool-chain is important due to technical and cultural differences between departments involved in the workflows of a large company, each with its own tooling and workflows.

2.12. D7.14 Demonstrator for Advanced Emergency Braking Systems

2.12.1. Objectives

The demonstrator shows an example of eFMI workflow through the Dassault Systèmes applications, especially focused on the continuity of successive activities.

The scenario is based on an **Advanced Emergency Braking System (AEBS) controller** development from Modelica design to AUTOSAR Adaptive Platform deployment, and towards the execution on the target.

The objectives of this demonstration are:

- to demonstrate the suitability of Modelica for block-diagram controller development,
- to provide a seamless integrated eFMI-based tool-chain (with continuous verification & validation of each step through each abstraction level),
- to prove the AUTOSAR compliance through the eFMI tool-chain.

In this demo, actors with different profiles do the successive activities.

Besides these points the goal is also to showcase the traceability and interoperability with different tools based on the eFMI workflow with the compliance of different standards and regulations (MISRA, SAFETY …).

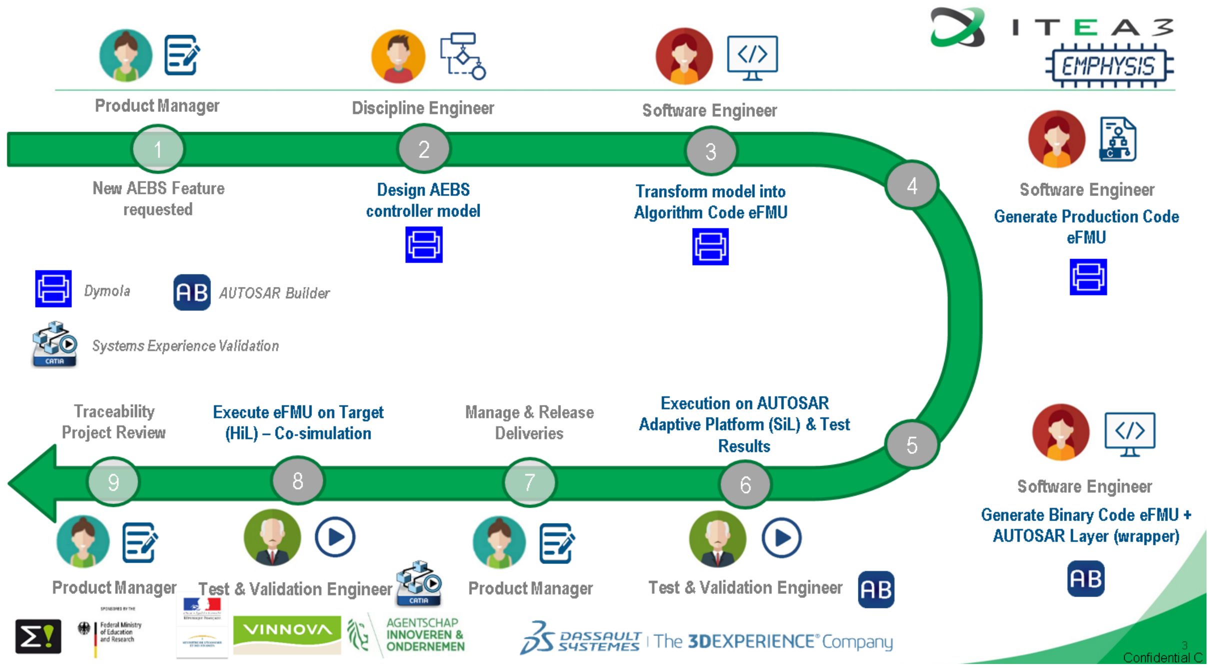

Product Manager
Discipline Engineer
Software Engineer
1
New AEBS Feature requested
2
Design AEBS controller model
3
Transform model into Algorithm Code eFMU
4
Software Engineer
Generate Production Code eFMU
Dymola
AUTOSAR Builder
Systems Experience Validation
Traceability Project Review
Execute eFMU on Target (HiL) – Co-simulation
Manage & Release Deliveries
Execution on AUTOSAR Adaptive Platform (SiL) & Test Results
5
6
7
8
9
Software Engineer
Generate Binary Code eFMU + AUTOSAR Layer (wrapper)
Product Manager
Test & Validation Engineer
Product Manager
Test & Validation Engineer
VINNOVA
AGENTSCHAP INNOVEREN & ONDERNEMEN
DASSAULT SYSTEMES | The 3DEXPERIENCE Company
Confidential C

Figure 27: Loop of the controller development in the eFMI workflow

2.12.2. Demonstrator description

The scenario that we have built is composed of 9 steps:

1- New AEBS feature requested

In the **3D**EXPERIENCE Platform, we start from a request to build this new feature at the product management level: requirements and tasks to implement.

2- Design AEBS controller model

In this step, we have modeled the AEBS controller through **Dymola** with causal design and enabled subsystems associated with signal locks. Before going further, we have also built the associated simulation model with clocked open-loop test scenarios in order to verify and validate it.

3- Transform model into *Algorithm Code* eFMU

After the validation of the AEBS Modelica model we start the *Algorithm Code* eFMU generation by configuring the Dymola's GALEC part with obfuscation level and the associated configuration (sampling period, solver ..) directly into the tool and build an *Algorithm Code* container that could be directly inspected with the GALEC code and manifest.

4- Transform model into *Production Code* eFMU

In the same way, we generate the *Production Code* eFMU by configuring the services from **ESP** application (Embedded Software Producer) and build it to introspect the generated container with MISRA-compliant C code and manifest file.

5- Generate *Binary Code* eFMU + AUTOSAR Layer (wrapper)

After obtaining the *Production Code*, we use **AUTOSAR Builder** to generate the associated AUTOSAR adaptive application. The *Binary Code* is generated through the AUTOSAR adaptive SDK.

6- Execution on AUTOSAR Adaptive Platform (SiL) & Test Results

We now check the generated *Binary Code* eFMU directly through the AUTOSAR Builder application, to validate it and generate associated test results automatically.

7- Manage & Release deliveries

Through this step, we come back into the **3D**EXPERIENCE Platform to consolidate all the work done. Each delivery can be captured through different ways and connected into the overall architecture through the Systems Traceability application. By connecting the deliveries, we add links between each component to see coverage analysis, implement links, and release the object when the work done is validated.

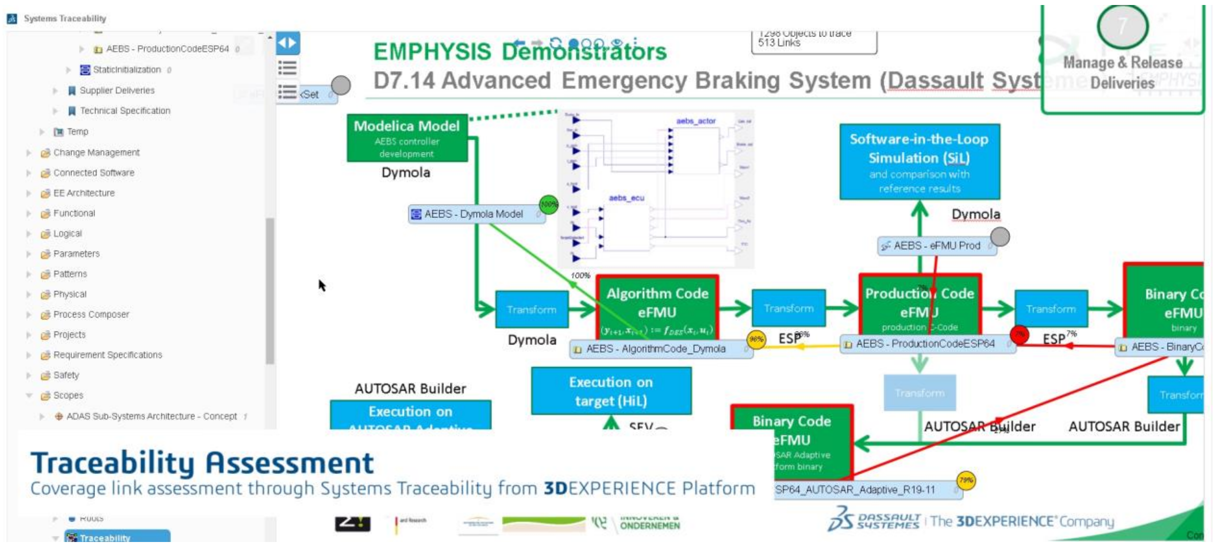

Systems Traceability
EMPHYSIS Demonstrators
D7.14 Advanced Emergency Braking System (Dassault Systèmes)
Manage & Release Deliveries
Modelica Model
AEBS controller development
Dymola
AEBS - Dymola Model
Software-in-the-Loop Simulation (SiL)
and comparison with reference results
Dymola
AEBS - eFMU Prod
Algorithm Code eFMU
Production Code eFMU
Binary Code eFMU
Transform
ESP
AEBS - AlgorithmCode_Dymola
AEBS - ProductionCodeESP64
AUTOSAR Builder
Execution on target (HiL)
Binary Code eFMU
Traceability Assessment
Coverage link assessment through Systems Traceability from 3DEXPERIENCE Platform
DASSAULT SYSTEMES | The 3DEXPERIENCE Company

Figure 28: Traceability and coverage levels along the eFMI workflow

8- Execute eFMU on target (HiL) – Co-simulation

With eFMU *Binary Code*, the final goal is now to execute it on a target for hardware in the loop simulation. In our case, we execute it on a Raspberry Pi and do a co-simulation with the **3D**EXPERIENCE Platform thanks to the **SEV (Systems Experience Validation**) **application**. We visualize the results with the associated model and through conceptual HMI to interact with the connected target.

In our case, to verify and validate that the correct behavior is obtained which is here the warning of a possible issue and the automatic emergency braking in appropriate cases according to the time-to-collision requirement.

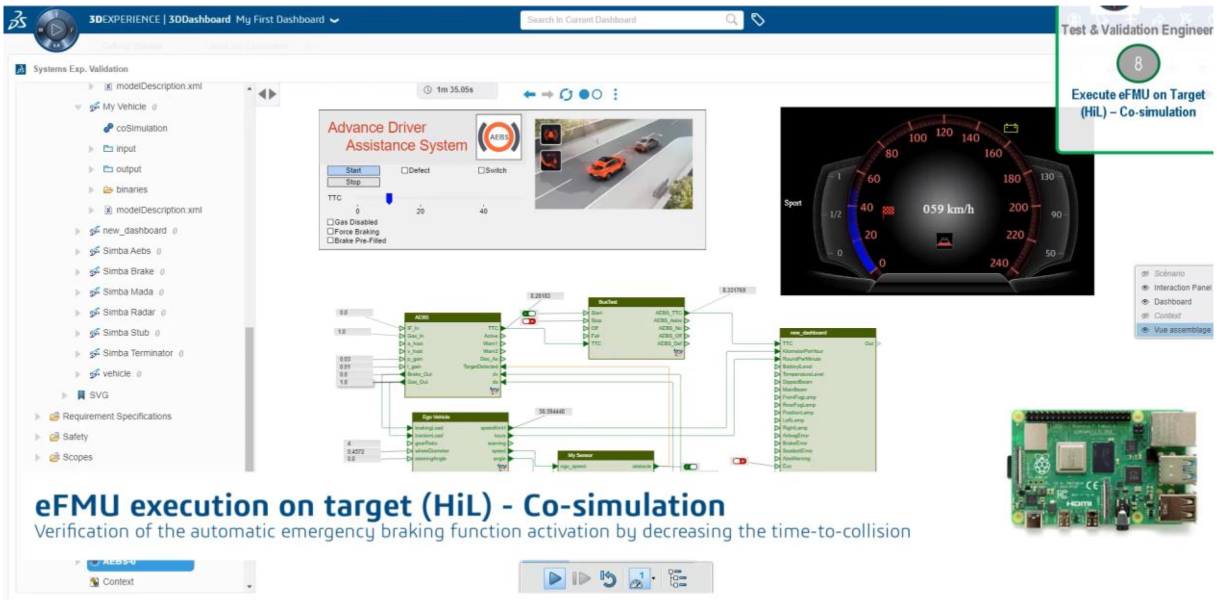

Test & Validation Engineer
8
Execute eFMU on Target (HiL) – Co-simulation
Advance Driver Assistance System
eFMU execution on target (HiL) - Co-simulation
Verification of the automatic emergency braking function activation by decreasing the time-to-collision

Figure 29: Controller execution with SEV

9- Traceability Project Review

To conclude on the project, we handle the traceability between the activities. **Systems Traceability** application performs traceability assessment and change impact analysis.

In that way, we are able to see if the components are connected along the project life cycle and if a change is needed, what will be impacted through the overall architecture.

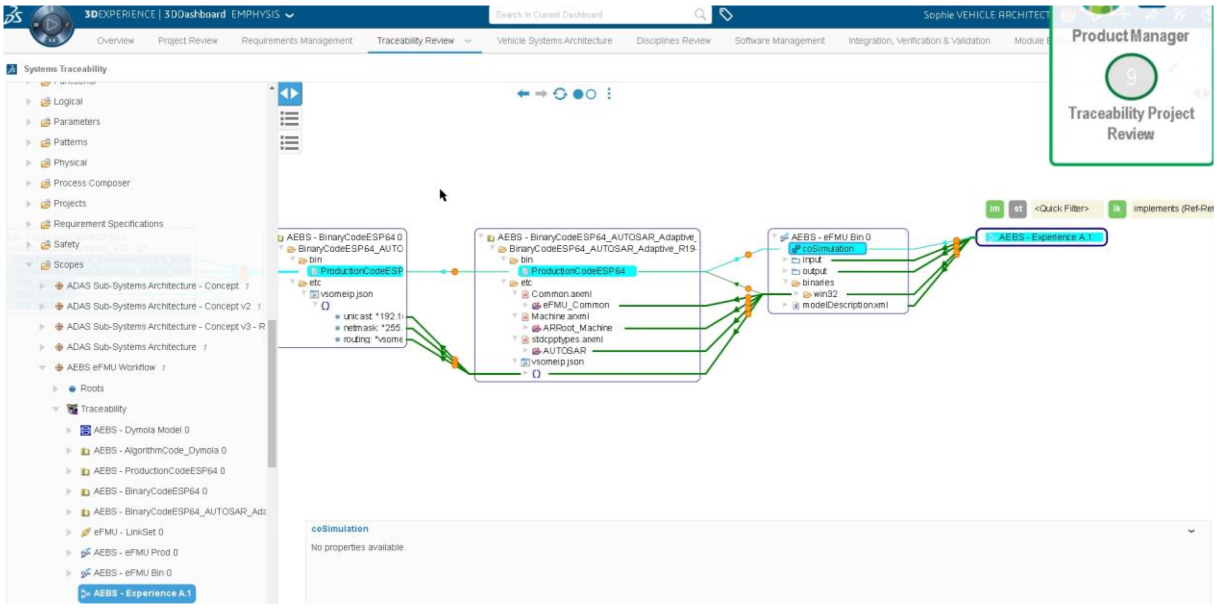


Figure 30: Detailed project traceability

2.12.3. Results summary

We have highlighted the block-diagram controller modelling in Dymola and showcased seamless transformation in an eFMI-based tool-chain from Modelica to AUTOSAR Adaptive Platform.

Transformation was done from Modelica model to *Binary Code*.

Checks and tests are possible along the successive steps of this transformation.

The demonstrator also shows the possibility to support eFMI in both standalone individual tools, and through the **3D**EXPERIENCE platform integration (first integration level is shown here).

The eFMI support will continue to mature the eFMI within tools and pursue the integration, for a fully supported eFMU workflow within the **3D**EXPERIENCE Platform.

3. Summary

This report illustrates the readiness of the developed eFMI specification for industrial applications and the high innovation potential for future applications in the automotive industry and beyond.

The presented test results prove the high maturity level of the developed tools and their interoperability as tool chain implementing the proposed eFMI workflow.

The variety of demonstrators illustrate the flexibility and versatility of the workflow. Challenging applications prove the developed technology to enable new and better ways of model-based development of embedded software.

4. References

[1] Lenord, Oliver, Martin Otter, Christoff Bürger, Michael Hussmann, Pierre Le Bihan, Jörg Niere, Andreas Pfeiffer, Robert Reicherdt, and Kai Werther. 2021. "eFMI: An Open Standard for Physical Models in Embedded Software." *Proceedings of the 14th International Modelica Conference*, Sept. 20-24, 2021. Linköping, Sweden. https://doi.org/10.3384/ecp2118157.

[2] ITEA3 Call2 Project 15016 EMPHYSIS – Embedded systems with physical models in the production code software https://itea3.org/project/emphysis.html.

[3] "EMPHYSIS - Functional Mock-up Interface for Embedded Systems." 2021. Accessed May 14. https://emphysis.github.io/.

[4] Armugham, Siva Sankar, Christian Bertsch, Oliver Lenord, and Kai Werther. 2021. "EFMI (FMI for Embedded Systems) in AUTOSAR for Next Generation Automotive Software Development." In Proceedings of the *17th Symposium on International Automotive Technology (SIAT)*, Sept. 29-Oct.1, 2021. Pune, India.

[5] Martin Sjölund, Oliver Lenord, Hans Olsson, Henrik Tidefelt, Kai Werther, and Adrian Pop (2021). "Towards a simplified form of Modelica without object-orientation". In *Software-Intensive Cyber-Physical Systems (SICS)* (planned).

[6] Rueger, Johannes-Joerg, Alexander Wernet, Hasan-Ferit Kececi, and Thomas Thiel. 2013. "MDG1: The New, Scalable, and Powerful ECU Platform from Bosch." In *Proceedings of the FISITA 2012 World Automotive Congress*, edited by SAE-China and FISITA, 194:417–25. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33829-8_39.