ITEA Review: EMPHYSIS #3

Introduction

Feb. 10, 2021 (13.00 – 17.00), Web-Meeting
EMPHYSIS Consortium

### Germany
- Bosch\(^1,3\)
- DLR\(^2\)
- ETAS
- ESI ITI
- AbsInt
- PikeTec
- dSPACE
- EFS

### Sweden
- Dassault Systèmes AB\(^3\)
- Volvo Cars
- Modelon
- Linköping University
- SICS East

### France
- Siemens SAS\(^3\)
- Dassault Systèmes SE
- Renault
- CEA
- University of Grenoble
- FH Electronics
- OSE
- Soben

### Belgium
- Siemens NV\(^3\)
- Dana
- University of Antwerp

### Canada*\(^1\)
- Maplesoft\(^3\)

### OEM Advisory Board
- BMW
- Daimler
- Mazda
- Volvo Trucks
- JSAE

<table>
<thead>
<tr>
<th></th>
<th>Large</th>
<th>SME</th>
<th>Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germany</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>France</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Sweden</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Belgium</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Canada*</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* w/o funding  
1) Project Lead  
2) Technical Coordination  
3) National Coordination
Project Overview
Project Overview
Bridge the gap

Modeling & Simulation

Embedded Software
Project Overview
Bridge the gap

Modeling & Simulation

Embedded Software
Online physical models key technology for advanced engine control software:

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

Physical models:

- Typically described by differential equations, best suited for dynamics
- Complementary to data-based modeling, can be combined
- Reduced calibration effort due to physical parameters
What is new?
State-of-the-art

Physical Modeling
(Domain Knowledge, Physical Principles & Phenomena, System Dynamics, Model Validation, ...)

Control Engineering
(System Theory, Stability, Robustness, ...)

Numerics
(Algorithms, Complexity, Stability, Precision, Realtime Performance...)

ECU Software
(MISRA, ASIL, MSR, AUTOSAR, ...)

Super Hero Function Developer
What is new?
New standard, new tool chains, new ways of collaboration
Project Overview

The eFMI workflow

- Input
- Output
- Residual generation
- Residual evaluation
- Knowledge of faults

Physical Model

Controller Model

Production Code

ECU Software

ECU Application
**Project Overview**

**Special requirements of automotive embedded systems**

- Specialized hardware: µController
  - Limited data memory and code memory, static memory allocation.
  - Single precision due to restricted data types (fixed-point, float).

- High safety requirements on the software:
  - Special coding guidelines, e.g., MISRA rules,
  - No exception handling (NaN, Division-by-zero,…),
  - Inbound guarantees.

- Hard realtime requirements on cyclic tasks:
  - Guaranteed execution time.
  - Limited smallest possible sampling rate (typically 1ms).

- Special realtime operating systems (AUTOSAR-OS)

- Specialized tools and tool chains (compilers etc.)
Special requirements of automotive embedded systems

- Specialized hardware: µController
  - Limited data memory and code memory,
  - Static memory allocation.
  - Single precision due to restricted data types (fixed-point, float).

- High safety requirements
  - Special coding guidelines, e.g., MISRA rules,
  - No exception handling (NaN, Division-by-zero, …),
  - Inbound guarantees.

- Hard realtime requirements
  - Guaranteed execution time,
  - Limited smallest possible sampling rate (typically 1ms).

- Special realtime operating systems
  - Specialized tools and tool chains (compilers etc.).

Project Overview

Bosch MDG1 ECU: current multi-core ECU

Motor Industry Software Reliability Association

AUTOSAR architecture
# Project Overview

## Business impact

### Increase Productivity
- Reuse
  - Model Libraries
  - Numerical Service Functions
- Automation
  - Model Transformations
  - Code Generation
- Seamless Tool Chain

### Master Complexity
- Software Design
  - Abstraction
  - Encapsulation
- Separation of Concerns
  - Physical Behavior
  - Data Flow
  - Embedded Code

### Software Innovations
- Tool Vendors
  - Added Value
  - Expand Market in MBD Domain
- Supplier/OEM
  - New Advanced Functions
  - Replace HW with SW
  - New Modes of Collaboration

---

### Physical Modeling & Simulation
- Component Libraries
- Physical Model

### ECU Software Development
- TargetLink
- ESP
- Production Code
- Services Functions
- RTPC, e.g., ETAS RTPC
- Bosch ECU
- Bosch VCU

---

Research & Development:
- ITEA3
- EMPHISIS
- VINNOVA
- Agentschap Innovatie & Ondernemen
Project Overview
Main Goals

- eFMI Standard
  - Exchange format from physical models to embedded software.
- eFMI Workflow → Tool Chain
  - eFMI supporting tools through all stages
- eFMI Demonstrators
  - Mature prototypes close to product release.
  - Better than state of the art performance.
  - Proven benefits for model-based control applications.
  - New innovative solutions enabled by eFMI.
  - New products, services, collaborations after project end.
Key Achievements
Key Achievements - Specification
eFMI Specification 1.0.0-alpha.3 available for public review:

https://emphysis.github.io/

Provided to FMI group in 2020 and incorporated their feedback.

Formal standardization process via the Modelica Association started.

Expected to be released as a Modelica Association standard in 2-3 months.
eFMI Specification Overview (chapter 3-6)

Chapter 3

Model (Modelica, Simulink, syq,...) → Simulations of Model

Typically a-causal, equation-based physics description (e.g., Modelica)

Behavioral Model eFMU
Reference results: \((t_i, u_i, y_i)\)

csv-files + XML manifest

Chapter 4

Algorithm Code eFMU
\(y_{i+1}, x_{i+1} = f_{DES}(x_i, u_i)\)

Transform

GALEC-code + XML manifest
Target-independent, algorithmic (i.e., causal) intermediate language for sampled-data systems

Chapter 5

Production Code eFMU
production C-Code + FMI for Co-Sim. C-wrapper

Transform

C-code + XML manifest

Testing of eFMI C-Code

Verification of eFMI C-Code

Software-in-the-Loop Simulation (SiL)

Chapter 6

Binary Code eFMU
PC binary + SOA app + target specific binary

Execution in Target Env. (compiled prod. C-Code)

Causal and acausal modeling tools

Abbreviations

eFMI: Functional Mockup Interface for Embedded systems
eFMU: Functional Mockup Unit for Embedded systems
GALEC: Guarded Algorithmic Language for Embedded Control

→ eFMI Standardization in order that tools can work seamlessly together

Typically a-causal, equation-based physics description (e.g., Modelica)

Algorithm Code eFMU

Transform

GALEC-code + XML manifest
Target-independent, algorithmic (i.e., causal) intermediate language for sampled-data systems

Production Code eFMU
production C-Code + FMI for Co-Sim. C-wrapper

Testing of eFMI C-Code

C-code + XML manifest

Verify of eFMI C-Code

Software-in-the-Loop Simulation (SiL)

Binary Code eFMU
PC binary + SOA app + target specific binary

Execution in Target Env. (compiled prod. C-Code)

Causal and acausal modeling tools

Abbreviations

eFMI: Functional Mockup Interface for Embedded systems
eFMU: Functional Mockup Unit for Embedded systems
GALEC: Guarded Algorithmic Language for Embedded Control

→ eFMI Standardization in order that tools can work seamlessly together

Typically a-causal, equation-based physics description (e.g., Modelica)
eFMU Specification
Container Architecture (chapter 2)

Different packaging formats: Example: FMU packed

Software-in-the-Loop-Simulation with every FMU tool

Classic Co-Simulation-FMU

eFMU Manifest

- Reference results
- Behavioral Model
- Manifest
- csv files

Algorithm Code
- Manifest
- GALEC code

Target independent

Production Code
- Manifest
- C-code

Target 1

Production Code
- Manifest
- C-code

Target N

Production Code
- Manifest
- C-code

Target 1

Binary Code
- Manifest
- Object code

Target M

Binary Code
- Manifest
- Object code

Manifest: Description of the interface of the associated code and additional meta information on how to access and utilize the code.

Model Description: Legacy meta information describing the model interface in the standard FMI format.
eFMI Specification Overview (chapter 3-6)

Chapter 3
Behavioral Model

\[
y_{i+1} = f_{DES}(x_i, u_i) 
\]

Reference results: \( (t_i, u_i, y_i) \)

csv-files + XML manifest

Chapter 4
Algorithm Code

\[
y_{i+2}, x_{i+1} := f_{DES}(x_i, u_i) 
\]

GALEC-code + XML manifest

Target-independent, algorithmic (i.e., causal) intermediate language for sampled-data systems

Chapter 5
Production Code

C-code + XML manifest

Testing of eFMI C-Code

Transform

Verification of eFMI C-Code

Chapter 6
Binary Code

object-code + XML manifest

Execution in Target Env. (compiled prod. C-Code)

Typical a-causal, equation-based physics description (e.g., Modelica)

modelling world

Transform

find causal, upper-bounded solution-algorithm

Abbreviations

eFMI: Functional Mockup Interface for Embedded systems
eFMU: Functional Mockup Unit for Embedded systems
GALEC: Guarded Algorithmic Language for Embedded Control

Causal and acasual modeling tools

Software-in-the-Loop Simulation (SiL)
eFMI Specification
Overview (chapter 3-6)

Chapter 3

Behavioral Model

eFMI Reference results: \((t_i, u_i, y_i)\)

csv-files + XML manifest

Chapter 4

Algorithm Code

eFMU

\(y_{i+1}, x_{i+1} := f_{DES}(x_i, u_i)\)

Transform

Chapter 5

Production Code

eFMU

C-code + XML manifest

Testing of eFMI C-Code

Verification of eFMI C-Code

Chapter 6

Binary Code

eFMU

object-code + XML manifest

Execution in Target Env.

(compiled prod. C-Code)

Causal and acausal modeling tools

Software-in-the-Loop Simulation (SiL)

Software-in-the-Loop Simulation (SiL)

Typical a-causal, equation-based physics description (e.g., Modelica)

find causal, upper-bounded solution-algorithm

GALEC-code

= guarded algorithmic language for embedded control
= program guarantees
= guards the following tool chain

Procedure

Abbreviations

eFMI: Functional Mockup Interface for Embedded systems
eFMU: Functional Mockup Unit for Embedded systems
GALEC: Guarded Algorithmic Language for Embedded Control
eFMI Specification
GALEC (chapter 3)

GALEC: Guarded Algorithmic Language for Embedded Control
- Target-independent, intermediate representation for bounded algorithms with multi-dimensional real arithmetics
- Imperative / causal language
- Safe – embedded & real-time suited – semantics
- Safe floating-point numerics
- Built-in mathematical functions
  (e.g. sin, cos, interpolation 1D & 2D, solve linear equation systems)

\[
x_{i+1} = f_x(x_i, u_i)
\]
\[
y_i = f_y(x_i, u_i)
\]

```plaintext
'gain.y' := self.gearRatio * self.wLoadRef;
'feedback.y' := 'gain.y' - self.wMotor;
'PID.D.y' := self.kd * ('feedback.y' - self.'PID.D.x') / self.Td;
'PID.y' := self.k * ('PID.D.y' + self.'PID.I.x' + 'feedback.y');
```
Safe – embedded & real-time suited – semantics
- Upper bound on number of operations
  ⇒ statically know resource requirements
  ⇒ exception free runtime semantic
- Statically known sizes (vectors, matrices etc.)
- Well-bounded indexing
  ⇒ never out-of-bounds / illegal memory accesses
- By value semantics with only well-defined & never competing side-effects
  ⇒ Huge potential for parallel execution (e.g., SIMD on multi-dimensions)

Language guarantees ⇒ satisfied by every GALEC program ⇒ following eFMI tool chain can leverage on
Safe floating-point numerics

- Ranged variables & implicit limitation at start/end of sample period
- Guaranteed qNaN (quiet-Not-a-Number) & error signal propagation
  + control-flow integrated error signal handling

⇒ No undetected errors
⇒ Enables back-up strategy at end of algorithm in case of any unexpected errors
Key Achievements - Tooling
The eFMI workflow is supported by these categories of tools

- Modeling & simulation tools (WP4)
- Embedded software tools (WP5)
- Verification & validation tools (WP6)

Tool prototyping running in parallel with the eFMI specification work from the start

- Cross-testing of tools in close collaboration between WP4/5/6
  - Test cases developed in WP7.1 were used to verify tool coverage
  - Several eFMI plug fests were organized for efficient interactions between tool vendors

- eFMI Compliance Checker developed to support tool implementations
Tool Prototypes
EMPHYSIS Workflow – Tool Positioning

Acausal tools in EMPHYSIS:
- Dymola (Dassault Systèmes)
- OpenModelica Compiler (LIU)
- SCODE-CONGRA (ETAS)
- SimulationX (ESI ITI)
- OPTIMICA (Modelon) – under dev.

Causal tools in EMPHYSIS:
- Amesim (Siemens SAS)
- Simulink (Mathworks)

Model in EMPHYSIS:
- Physical model (e.g. vehicle model)
- Controller, estimator, ...
- Diagnosis system, neural network, ...
- Any combination from above

Model in EMPYHSIS:
\[ 0 = f_{DAB}(x, \dot{x}, y, u) \]

Equation Code
\[ eFMU \]

\[ y_{i+1}, x_{i+1} \leftarrow f_{DES}(x_i, u_i) \]

Algorithm Code

\[ eFMU \]

\[ 0 = f_{DAE}(x, \dot{x}, y, u) \]

Production Code
\[ eFMU \]

\[ \text{eFMU code} + \text{FMI for Co-Sim. C-wrapper} \]

Binary Code
\[ eFMU \]

\[ \text{PC binary + SOA app + target specific binary} \]

Execution in Target Env.
\[ \text{(compiled prod. C-Code)} \]

Causal and acausal modeling tools

Software-in-the-Loop Simulation (SIL)

Testing of eFMI C-Code

ESP + AUTOSAR Builder (DS)

Astrée (AbsInt)
CSD (Siemens)

Further tool chains (not shown above):
- Physical model is transformed to eFMI Algorithm-Code which is used in a (controller/...) model from which eFMI Algorithm-Code is generated.
- Production C-Code may be directly generated from eFMI Algorithm Code.
- Embedded software architecture - design space exploration

Transform

TargetLink (dSPACE) ESP (DS)
SCODE-CONGRA (ETAS)

• inputs + outputs
• integrator
• interfaces of services functions

• generic or specific target configuration
• access of variables, services functions, ...

Transform

Transform

Transform

Transform

Transform

Transform

Transform
<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Vendor</th>
<th>EquCode</th>
<th>AlgCode</th>
<th>ProdCode</th>
<th>BinaryCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amesim</td>
<td>Siemens</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dymola</td>
<td>Dassault Systèmes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTIMICA Compiler Toolkit</td>
<td>Modelon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenModelica</td>
<td>Linköping University</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimulationX</td>
<td>ESI-ITI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCODE-CONGRA</td>
<td>ETAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUTOSAR Builder</td>
<td>Dassault Systèmes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TargetLink</td>
<td>dSPACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Astrée</td>
<td>AbsInt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSD</td>
<td>Siemens</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPT</td>
<td>PikeTec</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CATIA ESP</td>
<td>Dassault Systèmes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuaRTOS-DSE</td>
<td>CEA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Support for eFMI Import and Export

- Import: Prototype
- Export: Prototype
- Under dev. prototype
Tool Prototypes

eFMI Compliance Checker

- An open-source library for:
  - Verifying the eFMU architecture
  - Consistency checking of all model representation manifests
  - Validating the GALEC code against the specification

- Implemented in Python:
  - Fully documented
  - Easy to update and extend
  - Will be hosted on the Modelica Association Github repositories
  - Will be provided to the PyPI repository
Tool Prototypes

eFMI Compliance Checker

Test results and number of eFMUs that passed the compliance check:

<table>
<thead>
<tr>
<th>eFMUs vendor</th>
<th>Total</th>
<th>Consistency Check</th>
<th>GALEC code validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amesim</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Dymola</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>SimulationX</td>
<td>27</td>
<td>27</td>
<td>27</td>
</tr>
</tbody>
</table>

100%

KPI5: The eFMI compliance checker prototype (D6.1) does not report errors to at least 90% of the eFMI test components exported by all tool prototypes of WP4.
Key Achievements - Test cases
Test cases:
- Modelica library with 22 test cases containing 43 variants
- 3 Amesim models
- 2 manual GALEC codes

Features and Challenges:
- Inverse model or 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
eFMI toolchain for test cases:

- 9 commercial tools
- 50 toolchain paths
- Common GIT repository for eFMU-exchange and reports (~7 GB)
- 11 two-day plug fests
  - to test tool compatibility
  - to enhance eFMI specification

Diagram:
- 48 variants
- 67 eFMUs
- 201 eFMUs
- 206 eFMUs
- 402 analysis cases
- 538 test scenarios
- Modelica models
- Amesim models
- Manual GALEC code
- Dymola
- SimulationX
- Amesim
- Manual Implementation
- TargetLink
- Scode-Congra
- ESP
- Astrée
- TPT
- Analysis report
- Test report
- AUTOSAR Builder
Tool Prototypes
Test Cases and Test Coverage – Demonstrator D7.1

Test case coverage

- Algorithm code eFMU generation: 67
- Algorithm code eFMU validation (by 3 tools): 201
- Production code eFMU generation: 201
- Production code eFMU static code analysis: 102, 49, 7
- Production code eFMU test verification: 192, 9

 Passed | Passed after inspection | MISRA rule violations | Errors | Failed
Problem Statement:
- Embedded performance is crucial for user acceptance.

Objective
- Evaluation of the eFMI tool chain results against state-of-the-art embedded SW development.

Targeted Results
- KPI 8: Performance against state of the art (manual) implementation
  - At least 5 times less time to deliver model/controller function.
  - At most 25% less efficient code.
  - At most 25% more memory consumption.
    (25% increased overhead is acceptable due to the large increase in development efficiency and the expected increase in computing power and available memory in the next years.)
- KPI 11: Gain in productivity \([\Delta PY/PY \%]\)
  - Acceleration by >20%. 
Key Achievements - Performance Assessment
EMPHYSIS Demonstrators
D7.2 eFMI Performance Assessment (Bosch)

6 Benchmark Test Cases
(out of 22 EMPHYSIS Test Cases)

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Difficulty</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>M03</td>
<td>PID</td>
<td>low</td>
<td>Minimal footprint incl. saturated IOs</td>
</tr>
<tr>
<td>M04</td>
<td>Drivetrain</td>
<td>medium</td>
<td>Inverse linear physical model</td>
</tr>
<tr>
<td>M15</td>
<td>Air System</td>
<td>medium</td>
<td>Stiff ODE with delay operator</td>
</tr>
<tr>
<td>M10</td>
<td>Inverse Slider Crank</td>
<td>high</td>
<td>Inverse non-linear physical model (DAE Index-1)</td>
</tr>
<tr>
<td>M16</td>
<td>ROM</td>
<td>high</td>
<td>High dimensional maps, solve a large linear eq system</td>
</tr>
<tr>
<td>M14</td>
<td>Rectifier</td>
<td>high</td>
<td>Advanced symbolic transformation to compact ODE form</td>
</tr>
</tbody>
</table>

**Algorithm Code**

$$y_{i+1}, x_{i+1} = f_{DES}(x_i, u_i)$$

**Production Code**

eFMU production C-Code + FMI for Co-Sim. C-wrapper

**Transform**

1. Dymola (DS)
2. SimulationX (ESI-ITI)
3. ESP (DS)
4. SCODE-CONGRA (ETAS)
5. TargetLink (dSPACE)
6. Execution on Target (compiled prod. C-Code)

*Difficulty for an automated procedure to achieve same quality as manual implementation.
ECU Runtime Performance:

- In all cases the eFMI generated code is below the +25% KPI margin.
- In 5 of 6 examples an eFMI exists that outperforms the hand code.
- In average the best performing eFMUs are 26% faster than the hand code.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Difficulty</th>
<th>Relative ECU Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Average</td>
</tr>
<tr>
<td>M03</td>
<td>PID</td>
<td>low</td>
<td>-7%</td>
</tr>
<tr>
<td>M04</td>
<td>Drivetrain</td>
<td>medium</td>
<td>+9%</td>
</tr>
<tr>
<td>M15</td>
<td>Air System</td>
<td>medium</td>
<td>+38%</td>
</tr>
<tr>
<td>M10</td>
<td>Inverse Slider Crank</td>
<td>high</td>
<td>-65%</td>
</tr>
<tr>
<td>M16</td>
<td>ROM</td>
<td>high</td>
<td>+4%</td>
</tr>
<tr>
<td>M14</td>
<td>Rectifier</td>
<td>high</td>
<td>+3%</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td></td>
<td><strong>-3%</strong></td>
</tr>
</tbody>
</table>

*Difficulty for an automated procedure to achieve same quality as manual implementation.

**EMPHYSIS Demonstrators**

**D7.2 eFMI Performance Assessment (Bosch)**

**Graphical modeling:**
- high level of reuse
- component-oriented

**Textual modeling:**
- compact formulation

**M14 in both variants**
ECU Memory Consumption:
- In 3 of 6 cases an eFMU is below the +25% KPI w.r.t. code memory.
- In 4 of 6 cases an eFMU is below the +25% KPI w.r.t. data memory.
- In 4 of 6 cases an eFMU **outperforms** the hand code.
- In average the best performing eFMU requires
  - 39% more code memory
  - 9% less data memory
  than the hand code.
EMPHYSIS Demonstrators
D7.2 eFMI Performance Assessment (Bosch)

eFMI Productivity Gain

- Modeling
- Embedded Implementation
- Validation

Development Effort [h]

PID
M03

Manual 93%
eFMI 93%

Manual 92%
eFMI 92%

Manual 88%
eFMI 88%

Manual -13%
eFMI 13%

Manual 52%
eFMI 52%

Manual 18%
eFMI 18%

Manual 0%
eFMI 0%

Manual 20%
eFMI 20%

Manual 40%
eFMI 40%

Manual 60%
eFMI 60%

Manual 80%
eFMI 80%

Manual 100%
eFMI 100%

Manual 120%
eFMI 120%

Manual 140%
eFMI 140%

Development Effort [h]

- PID
- Drivetrain
- Inverse Slider Crank
- Rectifier
- Air System
- ROM

Development Effort [h]

- Manual
- eFMI

Development Effort [h]

- M03
- M04
- M10
- M14
- M15
- M16

Development Effort [h]
Conclusion

- Applications with component-oriented models (M03, M04, M10) show an eFMI development **productivity gain of ~90%**
  - with a **speed-up in runtime of ~40%**
  - and better or reasonable memory consumption.
- Textually implemented Modelica models (M14, M15, M16) show an eFMI development **productivity gain of ~20%**
  - and allow to provide solutions that outperform the hand coded solutions also for difficult systems.
- The configuration of the code generators allows to chose the best trade-off between runtime performance and memory consumption for the application.

*Better code, less effort!*
Key Achievements – Comprehensive Industrial Demonstrators
D7.10 Siemens Dana Demonstrator

Hybrid engine torque prediction using scale model Neural Network

D7.12 DLR Demonstrator

Semic-active damping controller with nonlinear inverse model and nonlinear Kalman filter

D7.06 Renault Demonstrator

Kalman Filter air filling estimation using scale model Neural Network predictor

D7.08 Daimler Demonstrator

Dual-Clutch Transmission Diagnosis
Seamless production code build process with multiple step implicit integrator

D7.13 Volvo Demonstrator

Transmission model as virtual sensor

D7.3 Powertrain Vibration Reduction

Advanced Emergency Braking System controller

D7.4 Model-based Diagnosis of Thermo System

Semic-active damping controller with nonlinear inverse model and nonlinear Kalman filter

D7.07 GipsaLab

Vehicle dynamics control by Parameterized Nonlinear Model Predictive Control for semi-active control with Neural Network prediction model

D7.14 Dassault Systèmes Demonstrator

Advanced Emergency Braking System controller

D7.2 eFMI Performance Assessment

Vehicle dynamics control by Parameterized Nonlinear Model Predictive Control for semi-active control with Neural Network prediction model
Demonstrators Presentations

**D7.14 Dassault Systèmes**
Advanced Emergency Braking System controller

**D7.10 Siemens Dana Demonstrator**
Hybrid engine torque prediction using scale Neural Network model

**D7.05 Renault**
Kalman Filter TDC air filling estimation using scale Neural Network predictor model

**D7.06 Renault**
Throttle high frequency position estimation using scale NN predictor model

**D7.3 Powertrain Vibration Reduction**
Powertrain Vibration Reduction

**D7.4 Model-based Diagnosis of Thermo System**
Model-based Diagnosis of Thermo System

**D7.12 DLR**
Semic-active damping controller with nonlinear inverse model and nonlinear Kalman filter

**D7.08 Daimler**
Dual-Clutch Transmission Diagnosis using multiple step implicit integrator

**D7.14 Dassault Systèmes**
Advanced Emergency Braking System controller

**D7.07 GipsaLab**
Vehicle dynamics pNMPC for semi-active control with Neural Network prediction model

**D7.13 Volvo**
Transmission model as virtual sensor

**D7.2 eFMI Performance Assessment**
Six different use cases from simple PID control to complex physical ODE model
EMPHYSIS Key Achievements
Summary

- eFMI Specification – Alpha version published
- Formal standardization process started via the Modelica Association
  Expected to be released as a Modelica Association standard within 2-3 months
- Open-source Modelica library EMPHYSIS_TestCases to be released by end of Feb.
  Facilitating qualified cross-checking of the toolchain
- 13 tools are currently supporting different parts of the eFMI standard
- eFMI Compliance Checker available to support further adoption of the standard
- Extensive test library with 22 test cases containing 43 variants
- Excellent performance results
- Comprehensive industrial demonstrators of varying eFMI application scenarios
Synopsis
Synopsis
Main Goals

- eFMI Standard
  - Exchange format from physical models to embedded software.

- eFMI Workflow → Tool Chain
  - eFMI supporting tools through all stages

- eFMI Demonstrators
  - Mature prototypes close to product release.
  - Better than state of the art performance.
  - Proven benefits for model-based control applications.
  - New innovative solutions enabled by eFMI.
  - New products, services, collaborations after project end.

The journey has just begun!