Standardizing eFMI for Embedded Systems with Physical Models in the Production Code Software

What is this all about?
The new eFMI standard

- Why are you doing this? – Purpose/Motivation
- What is new? – Problem Statement/Benefit
- How does it work? – Conceptual Idea
- How does it work in practice? – Demonstrator
- How good does it work? – Performance Metrics
- Who will use it? – Usage Scenarios
- Who supports it? – Tool Prototypes

- When can I have it? – Project Schedule
- Who is doing all this? - Acknowledgements

Jubilee Symposium: Future Directions of System Modeling and Simulation
Sept. 30, 2019, Medicon Village, Lund, Sweden

Oliver Lenord, Robert Bosch GmbH – Corporate Research
with contributions from all partners
Why are you doing this?
Bridge the gap

Modeling & Simulation

Embedded Software

Jubilee Symposium 2019: Future Directions of System Modeling and Simulation
Physical Model

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

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

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

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

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

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

- Specialized hardware: µController with specialized cores, limitations in memory and data types (fixed-point, float)
- High safety requirements on the software
- Special coding guidelines, e.g., MISRA rules
- Special real-time operating systems (AUTOSAR-OS)
- Specialized tools and tool chains (compilers etc.)
- AUTOSAR standard defining the structure and interface of software modules, replacing proprietary solutions; support for some basic numerical functions

Today ECU software requirements are not satisfied by the FMI standard.
How does it work?
The eFMI workflow

Physical Model

Controller Model

Production Code

ECU Application

Model

(Modelica, Simulink, ...)

Acausal/causal tools

Acausal tools

Equation Code

eFMU

$\dot{\mathbf{x}} = f_{\text{eq}}(t, \mathbf{x}, \mathbf{u})$

Algorithm Code

eFMU

$\mathbf{y} = f_{\text{alg}}(t, \mathbf{x}, \mathbf{u})$

Software-in-the-Loop Simulation (SiL)

and comparison with reference results

Testing of eFMI C-Code

Verification of eFMI C-Code

Production Code

eFMU

production C-Code + eFMU for Co-Sim. Wrapper

Binary Code

eFMU

PC binary + Sim app + target specific binary

Execution on Target

compiled prod. C-Code
How does it work?
The eFMI enabled tool chains

Model (Modelica, Simulink, ...)

Acausal/causal tools

Transform

Equation Code eFMU

Acausal tools

Transform

Algorithm Code eFMU

Transform

Production Code eFMU

Software-in-the-Loop Simulation (SIL) and comparison with reference results

Testing of eFMI C-Code

Verification of eFMI C-Code

Binary Code eFMU

Execution on Target (compiled prod. C-Code)

Production Code eFMU

PC binary + SDA app + target specific binary

Execution on Target (compiled prod. C-Code)

Equation Code eFMU

Acausal/causal tools

Transform

Algorithm Code eFMU

Transform

Production Code eFMU

Software-in-the-Loop Simulation (SIL) and comparison with reference results

Testing of eFMI C-Code

Verification of eFMI C-Code

Binary Code eFMU

Execution on Target (compiled prod. C-Code)

Production Code eFMU

PC binary + SDA app + target specific binary

Execution on Target (compiled prod. C-Code)
How does it work?
The eFMI enabled tool chains

Model
(Modelica, Simulink, ...)
Acausal/causal tools
Acausal tools
Transform
Equation Code
eFMU
\[ \Phi = f_{\text{equ}}(x, y, w) \]
Algorithm Code
eFMU
\[ \Phi, \dot{x}(x, y, w) = f_{\text{alg}}(x, y, w) \]
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)

Software-in-the-Loop Simulation (SIL) and comparison with reference results
Testing of eFMI C-Code
Verification of eFMI C-Code

How does it work?
The eFMI container architecture

eFMU:
Substructure within FMU container.
eFMU Manifest:
Description of the available model representations and how to access them. Other general meta information.
Model Representation:
Compound of Code + Code Manifest representing the model in one particular standardized form.
Algorithm Code Manifest
Eq Code Manifest
Code Manifest:
Description of the model 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.

Jubilee Symposium 2019: Future Directions of System Modeling and Simulation
How does it work?
The eFMI Equation Code model representation

Definition:
- Flat unsorted set of equations (DAE).

Purpose:
- Capture the mathematical essence of a physical model.
- Exchange of models on equation level:
  - Enable advanced analysis methods, e.g. diagnosability analysis with SCODE CONGRA (ETAS).
  - Higher simulation efficiency and robustness compared to FMI based model exchange.
- Full back-traceability from to the original mathematical model.

Equation Code

New flat_model
- constant, parameter, discrete, enumeration, record, initial, equation, der, operator, function, pure, impure, return, input, output, external, true, false, and, or, not, if, then, else, elseif, when, elsewhen, annotation, end

Required
- final, flow, stream, type, class, block, protected, public

Dispensable
- algorithm, encapsulated, expandable, for, in, loop, while, break, each

Specification:
- Reference to future standardized Flattened Modelica:
  - No object-orientation.
  - No algorithms.

Level of Maturity: Low
- First proposal for Flattened Modelica to be discussed in the Modelica Language Group at the Modelica Design Meeting (Oct. 2019).

How does it work?
The eFMI Equation Code model representation

J1.phi=01.phi.rel+fixed.phi0;
S1.phi.rel=S2.phi-J1.phi;
S1.b.tau=S1.c*S1.phi.rel+S1.c*S1.phi.rel0;
J2.J*der(J2.w)=S1.b.tau;
D1.b.tau=D1.d*der(D1.phi.rel);
J1.b.tau=S1.b.tau-D1.b.tau;
J1.J*der(J1.w)=S1.b.tau-T1.tau;
**How does it work?**

**The eFMI Algorithm Code model representation**

**Definition:**
- Sampled input/output block
- Sorted set of assignments.
- Target independent "logical" representation of the Production Code.

**Purpose:**
- Representation of simple and advanced observers, diagnosis functions, health monitoring, controllers (inverse models, model predictive control, gain scheduling, extended Kalman filter, ...)
- Reuse of the same Algorithm for different constraints, targets and applications.
- Separation of concerns: Symbolic transformation vs. embedded code generation.

**How does it work?**

**The eFMI Algorithm Code model representation**

**Specification:**
- Standardized small subset of Modelica:
  - No equations, No inheritance.
  - Arrays with literal dimensions + operations, no dynamic memory allocation.
  - Statically guaranteed array access.
- with extensions:
  - Methods (DoStep, Initialize, ...)
  - Includes discretized integrator, such as explicit Euler or linear implicit Euler
  - Built-in functions for sin, cos, tables (1D, 2D, 3D), „solve linear equation system”, ...
  - Error handling.
- ...  

**Level of Maturity: Medium-High**
- Released first draft being used for on-going tool prototypes.
- Some simplifying assumptions to speed-up prototype development.
Definition:
- C code, compliant with coding standards (e.g. MISRA)
- Co-existing generic or optimized code for specific architectures (e.g. AUTOSAR) or targets.

Purpose:
- Best performance for dedicated target.
- Optimized resource demand (memory, CPU time).
- Seamless integration in ECU environment (no wrapper).
- Integration in Software-in-the-Loop testing tools.
- Enable code verification and compliance checks.

How does it work?
The eFMI Production Code model representation

void DoStep(eFMI *ctx, eFMI *eFMI)
{
    ...;
    _J1_phi_rel=D1_phi_rel+fixed_phi0;
    _S1_c=(_J1_c*_S1_phi_rel+_fixed_phi0);
    _J2_w=(_S1_b_tau)/_J2_J;
    _J1_b_tau=(_S1_b_tau-der__J1_w_rel)/_J1_J;
    _der__J1_w_rel=(_J1_b_tau+_T1_tau)/_J1_J;
}

Specification:
- No standardized API.
- Existence of the methods must be guaranteed.
- Interface and structure of the functions as described in the manifest file.
- May contain target specific code (e.g. assembler code).

Level of Maturity: High
- Released first draft being used for on-going tool prototypes.
**How does it work?**

The eFMI Binary Code model representation

**Definition:**
- Target specific object code
  (e.g. PC, ECU, Service oriented Architecture (SOA) applications)

**Purpose:**
- Seamless integration with other ECU software to build an image for a specific ECU.
- Separation in dedicated modules.
- Protection of IP.
- Protection of integrity.
- Enable execution of the **exact same code** on a test platform.

**How does it work?**

The eFMI Binary Code model representation

**Specification:**
- Binary object files and libraries, e.g. ELF files.
- Measurement, Calibration or Diagnostics Description files, e.g. A2L files
- Map file
- Linker file

**Level of Maturity:** medium
- Released first draft being used for on-going tool prototypes.
How does it work in practice?

**eFMI Tool Chain applied to Speed Controller example**

**Modelica**

- Model (Modelica, Simulink,...)

**Equation Code**

- \( \frac{d\text{Model}}{dt} = f(x, y, u) \)

**Transform**

- Dymola (DS)
- SimulationX (ESI-ITI)

**Algorithm Code**

- eFMI
  - TargetLink (dSPACE)
  - SCODE-CONGRA (ETAS)

**Transformation**

- Target-independent intermediate code
- Target-dependent C-code

**Testing of eFMI Code**

- Verification against rules and standards
- Testing of eFMI C-Code

**Binary Code**

- eFMI PC binary + SOA app + target specific library

**Execution on Target**

- Compiled production C-Code

**Performance Benchmark**

- Bosch ECU
How does it work in practice?

eFMI Tool Chain applied to Speed Controller example

- Modelica
- eFMU Manifest
- AlgCode
- Manifest
- ProdCode
- eFMU

Dymola

SimulationX

Whole system model with plant for testing the controller

Instance of the control model in the context of a clocked system (automatically discretized)

- Bosch ECU
- Astrée
- Dymola
- SimulationX

Export Dialog

SCODE-CONGRA

Export AlgCode

Jubilee Symposium 2019: Future Directions of System Modeling and Simulation
How does it work in practice?

eFMI Tool Chain applied to Speed Controller example

Modelica

- SpeedController

Export

AlgCode

Dymola

SimulationX

eFMU

- eFMU Manifest
- AlgCode
- Manifest
- AlgCode
- ProdCode
- Manifest
- ProdCode

eFMU

Bosch ECU

Astrée

Dymola

SimulationX

SCOPE-CONGRA

Content of the eFMU

- Unique
- Extendable
- Flexible
- Consistent
- Traceable
How does it work in practice?

**eFMI Tool Chain applied to Speed Controller example**

**Alg Manifest**
- Model Description with few extension
- Consistent
- Traceable
- Extendable

```plaintext
method doStep
protected
  Real gain_y;
  Real feedback_y;
  Real PI_x.der;
  Real PI.y;
algorithm
  gain_y := eFMU.gearRatio*eFMU.wLoadRef;
  feedback_y := gain_y - eFMU.wMotor;
  PI.x.der := feedback_y / eFMU.T;
  eFMU.PI.x := eFMU.PI.x + eFMU.T_sample*PI.x.der;
  PI.y := eFMU.k*(eFMU.PI.x + feedback_y);
  eFMU.wMotor := if PI.y > eFMU.limiter.uMax then eFMU.limiter.uMax else PI.y < eFMU.limiter.uMin then eFMU.limiter.uMin else PI.y;
end doStep;
```

**AlgCode**
- readable
- understandable
- discretized
- target independent
How does it work in practice?

eFMI Tool Chain applied to Speed Controller example

Modelica speedController

Imported AlgCode

Dymola

SimulinkX

Generated Prod Manifest
- consistent
- traceable
- flexible

Imported AlgCode

Generated ProdCode
- readable
- understandable
- target dependent
- optimized

Jubilee Symposium 2019: Future Directions of System Modeling and Simulation
How does it work in practice?
eFMI Tool Chain applied to Speed Controller example

- Import ProdCode FMU
  - back-to-back testing
  - generate/execute test cases

- ProdCode FMU
- ProdCode Manifest
- AlgCode
- AlgCode Manifest
- eFMU Manifest
- eFMU

- Static code analysis
  - RuleChecker to find rule violations (MISRA et. al.)

Jubilee Symposium 2019: Future Directions of System Modeling and Simulation
How does it work in practice?

**eFMI Tool Chain applied to Speed Controller example**

Integrate ProdCode in ECU SW Test Environment:
- Build SW
- Flash on ECU
- Verify results
- Measure CPU time
- Measure memory demand

<table>
<thead>
<tr>
<th>#</th>
<th>Test Case Name</th>
<th>Multi-Dim Maps</th>
<th>Large Maps</th>
<th>Large Matrices</th>
<th>Sparse Matrices</th>
<th>Nonlinear</th>
<th>Large Number of States</th>
<th>Compact Code</th>
<th>Stiff</th>
<th>DAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>M03</td>
<td>SpeedController</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

How good does it work?

**Performance benchmark**

<table>
<thead>
<tr>
<th>ProdCode Source</th>
<th>AlgCode Source</th>
<th>CPU Time</th>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand coded</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TargetLink</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCODE-CONGRA</td>
<td></td>
<td>First measurements available but not yet verified.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Manual discretization**

Auto-generated C code

Bosch ECU

Benchmark
Who will use it?
Usage scenarios and demonstrators

Diversity of applications
- Engine richness
- Engine vibrations
- Fault detection (thermal, cooling)
- After treatment
- Vehicle dynamics
- Energy Monitoring
- Torque vectoring
- Active damping

Control strategies versatility
- Feed forward
- Estimators
- Model Predictive Controls
- Non-Linear Model Predictive Controls
- Linear parameter-varying
- Kalman filters

Models types Versatility
- Non-linear models
- Inverted non-linear models
- Residuals model
- Linearized models
- Neural Networks

Tool independent format
- No S-Function constraint

Who will support it?
Planned and on-going tool development

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Equation Code</th>
<th>Algorithm Code</th>
<th>Production Code</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siemens – AMESim</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Dassault Systèmes – Dymola</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Modelon – JModelica</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenModelica</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI-ITI – SimulationX</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETAS – SCODE-CONGRA</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dassault Systèmes – AUTOSAR Builder</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dSPACE – TargetLink</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AbsInt – Astrée</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Siemens – CSD</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PikeTec – TPT</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
When can I have it?

Schedule

eFMI Supporting Tools
- EMPHYSIS Project ends by February 2021
  - All planned tool prototypes will be finalized.
  - Product readiness is expected not before mid 2021

eFMI Standard
- First draft has been finalized Mar. 2019
- After AlgCode and ProdCode have reached a stable state a preliminary version of the specification is considered to be shared under NDA.
- First official release after the end of the project after consultation of the Modelica Association.

Who is doing all this?

Acknowledgements

- Germany
  - Bosch\textsuperscript{1,3}
  - DLR\textsuperscript{2}
  - ETAS
  - ESI ITI
  - Abalint
  - PikeTec
  - dSPACE
  - EFS

- Sweden
  - Dassault Systèmes AB\textsuperscript{3}
  - Volvo Cars
  - Modelon
  - Linköping University
  - SICS East

- France
  - Siemens SAS\textsuperscript{3}
  - Dassault Systèmes SE
  - Renault
  - CEA
  - University of Grenoble
  - FH Electronics
  - OSE
  - Soben

- Belgium
  - Siemens NV\textsuperscript{2}
  - Dana
  - University of Antwerp

- Canada\textsuperscript{*}
  - Maplesoft\textsuperscript{3}

- OEM Advisory Board
  - BMW
  - Daimler
  - Mazda
  - Volvo

Special thanks to the highly engaged members of the first eFMI Plug Fest, Sept. 23-25, 2019, Renningen, Germany: Christoff Bürger, Kai Werther, Robert Reicherd, Reinhold Heckmann, Jörg Niere, Gerd Kurzbach, Jishnu Jayaram, Yuri Durodie, Martin Otter, Andreas Pfeiffer, Christian Bertsch, Oliver Lenord among many other contributors.

\textsuperscript{1) Project Lead}
\textsuperscript{2) Technical Coordination}
\textsuperscript{3) National Coordination}