Modelica Association

efmi Functional Mock-up Interface for embedded systems

eFMI® scope and delimitation

FMI User Meeting – 15th International Modelica Conference – 10th of October 2023

Christoff Bürger
Dassault Systèmes
Christoff.Buerger@3ds.com

Agenda

1. Scope of eFMI®: GALEC as example of satisfying non-functional quality requirements
2. Delimitation in embedded software domain: eFMI® vs. FMI®, AUTOSAR, ASAM, …

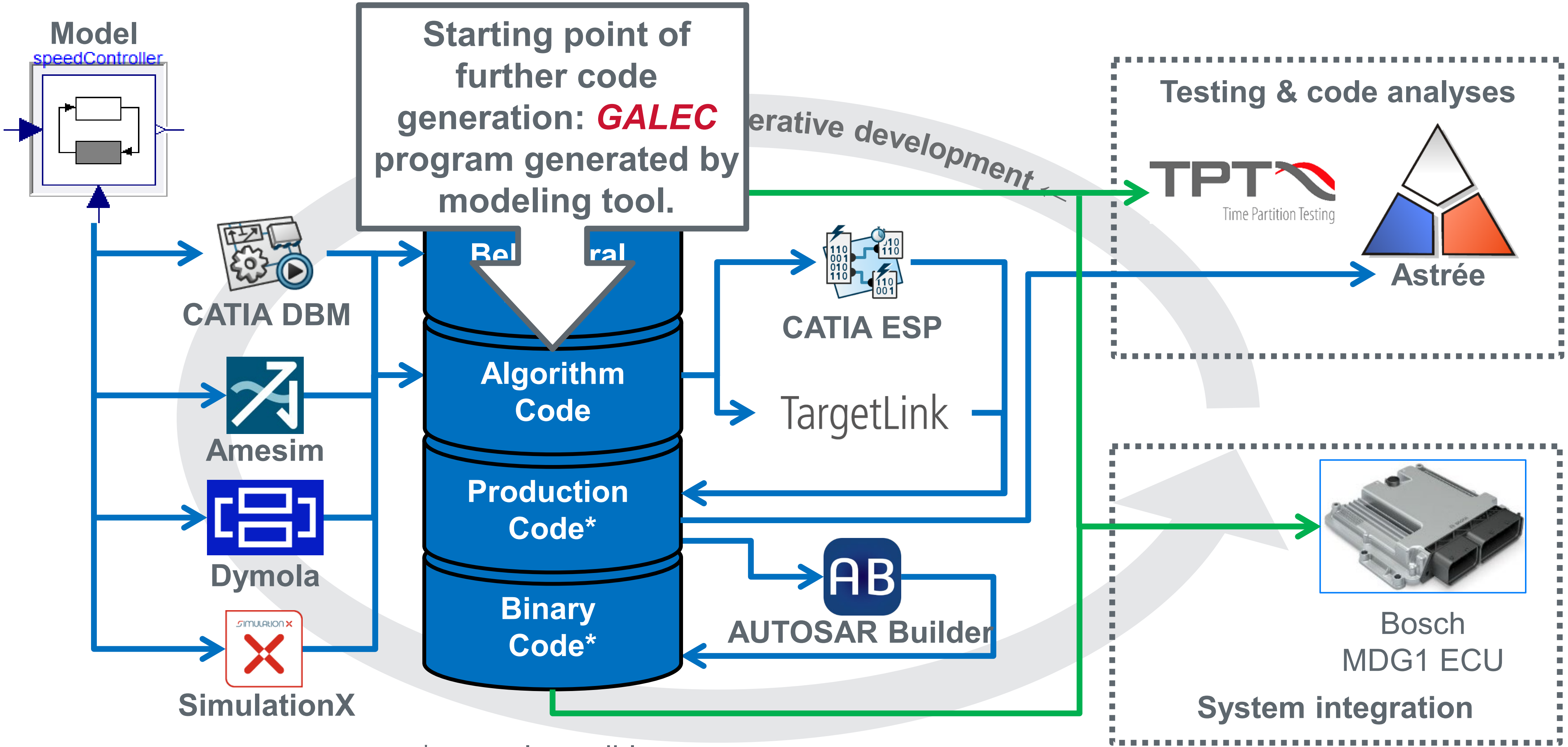
eFMI is all about:

How to *develop software satisfying non-functional requirements* besides just functional?

As an example, let us have a short look on eFMI GALEC.

(other examples would be eFMI Behavioral Models or inter-container linking for traceability)

eFMI Standard: Toolchain & workflow

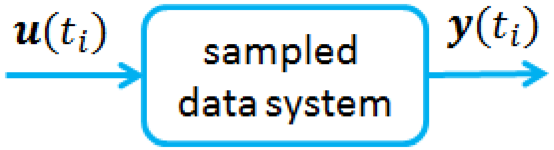


*several possible

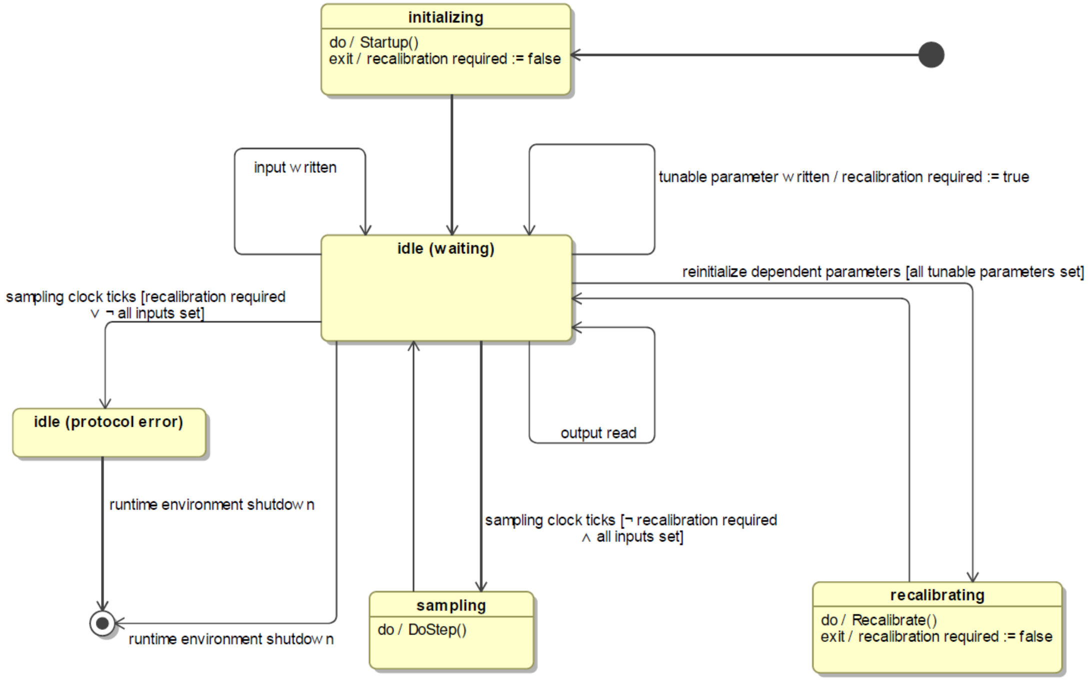
eFMI GALEC: Scope

GALEC (***G**uarded **A**lgorithmic **L**anguage for **E**mbedded **C**ontrol*): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation

GALEC program: sampled algorithm with fixed sampling period.



$$x_{i+1} = f_x(x_i, u_i)$$
$$y_i = f_y(x_i, u_i)$$



Block life-cycle specifies usage via common interface:

- (default) initialization
- sampling
- recalibration
- reinitialization

⇒ Defines valid system integration scenarios.

Functional Mock-up Interface for embedded systems

eFMI GALEC: Language characteristics

GALEC (***G**uarded **A**lgorithmic **L**anguage for **E**mbedded **C**ontrol*): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation

- Imperative / causal language of high abstraction level (e.g., multi-dimensional real arithmetic, built-in mathematical functions like sinus, cosine, interpolation 1D & 2D, solve linear equation systems etc.)
- Safe – embedded & real-time suited – and well-defined semantics
  - Upper bound
  - Statically known sizes and safe indexing
  - Well-defined & never competing side effects
- Safe floating-point numerics
  - Guaranteed NaN propagation
  - Saturation of ranged variables
- Ordinary control-flow integrated, strict error handling concept
  - Guaranteed error signal propagation enables delayed error handling

⇒ Guards further eFMI tooling

efmi Functional Mock-up Interface for embedded systems

eFMI GALEC: Language characteristics

GALEC (***G**uarded **A**lgorithmic **L**anguage for **E**mbedded **C**ontrol*): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation

Imperative / causal language of high abstraction level:

- Target machine characteristics abstracted in:
  - Idealized types (Boolean, Integer & Real)
  - Builtin functions (e.g., construct & check NaN or ∞, convert Real ↔ Integer, extract fractional, rounding)

  ⇒ Idealized, but executable algorithms (math algorithms on computers)
- Builtin operators for multi-dimensional real arithmetic & builtin functions encapsulating common mathematical algorithms (e.g., interpolation 1D, 2D, 3D; solve linear equations)

  ⇒ Optimization for target environment at production code generation

eFMI GALEC: Language characteristics

GALEC (***G***uarded ***A***lgorithmic ***L***anguage for ***E***mbedded ***C***ontrol): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation

Imperative / causal language of high abstraction level:

- Well-defined onion-layered initialization:
  - Dependencies: constants ← tuneable parameters ← dependent parameters ← inputs ← states & outputs
  - Each has separate *algorithmic* initialization function

  ⇒ Safe, complex and optimizable initialization
- Simple block life cycle with support for input-dependent initialization, reinitialization & recalibration

Functional Mock-up Interface for embedded systems

eFMI GALEC: Language characteristics

GALEC (***G***uarded ***A***lgorithmic ***L***anguage for ***E***mbedded ***C***ontrol): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation

Imperative / causal language of high abstraction level:

- Safety & simplicity first:
  - Only for-loops and if-elseif-else control-flow
  - Only Integer, no, int, short, unsigned, long long etc
  - No implicit type conversions
  - Unique way to write Real literals: X.X[e(+|-)X] (not 1e10, 1E+10, 1.0e10, .0)
  - Only LF line endings, only UTF-8 encoding (code ASCII, comments UTF-8)
  - …

eFMI GALEC: Language characteristics

GALEC (***G**uarded **A**lgorithmic **L**anguage for **E**mbedded **C**ontrol*): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation.

Safe – embedded & real-time suited – and well-defined semantics:

- Statically known sizes and safe indexing:
  - No pointer arithmetic
  - No memory-layout implications for multi-dimensionals (like vector elements must be consecutive memory)
    ⇒ Production code generators can rearrange (e.g., scalarize & decompose) multi-dimensionals
  - Clear separation of statically-evaluable and run-time expressions; same syntax, but different evaluation times
    ⇒ Complex indexing expressions including, e.g., function calls, supported
  - Dependent dimensionalities (e.g., input must be square matrix, vector twice length of 1st dimension of matrix)
- Upper bound:
  - No recursion, only statically known looping (over size-fixed multi-dimensionals)
    ⇒ GALEC programs can be unrolled to sequence of conditional assignments.

efmi Functional Mock-up Interface for embedded systems

eFMI GALEC: Language characteristics

GALEC (***G***uarded ***A***lgorithmic ***L***anguage for ***E***mbedded ***C***ontrol): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation.

Safe – embedded & real-time suited – and well-defined semantics:

- Well-defined & never competing side effects
  - Unique access to global state (`self.name`)
  - Clear separation of functions (no access to global state) vs. methods (access to global state)
  - Fixed evaluation order of function/method arguments (left-to-right)
  - No method calls in argument-expressions
  - No aliases, only call by value, inputs cannot be assigned

⇒ For every two GALEC statements, it is decidable if they can be switched (automatic parallelization).

Functional Mock-up Interface for embedded systems

eFMI GALEC: Language characteristics

GALEC (***G***uarded ***A***lgorithmic ***L***anguage for ***E***mbedded ***C***ontrol): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation.

Safe floating-point numerics & ordinary control-flow integrated, strict error handling concept :

- Errors must be either handled in ordinary if-statements or propagated
  - Operations that can cause NaN signal errors (e.g., relational operators like <, <=, >, >=)
  - Signaled errors can be checked at later if-statements
    ⇒ delayed error handling (not C style spaghetti code on machine flags after each and every operation)
- Builtin functions signal errors:
  - Every builtin function when undefined either, propagates NaN as result or signals NaN error
  - Predefined signals for singular or non-unique linear equation systems, size issues (convert Real ↔ Integer) etc

⇒ Errors are always recognized (nothing slips through).

⇒ Enables handling of *unforeseen* runtime errors, for example, using a backup controller, reset to previous state etc.

efmi Functional Mock-up Interface for embedded systems

eFMI GALEC: Summary

GALEC (***G***uarded ***A***lgorithmic ***L***anguage for ***E***mbedded ***C***ontrol): Intermediate representation well-suited as code generation target for modelling tools & source for embedded-code generation

⇒ GALEC is by language design safe and guards further eFMI tooling.

- Not an (operating) system level programming language
  (that needs to be tamed by plethora of further anlyses tooling; pun on C & Co. intended)
- Production code tooling can optimize code – thanks to GALEC guarantees – by lowering abstraction
  (which need no artificial taming, but can be if required, e.g., MISRA C:2012 compliance)

⇒ Simple language with well-defined semantic, well-suited for expressing and long term archiving algorithmic solutions of physics models.

⇒ A language for safety-critical and real-time suited (control-)algorithms.

Functional Mock-up Interface for embedded systems

Agenda

1. Scope of eFMI®: GALEC as example of satisfying non-functional quality requirements
2. Delimitation in embedded software domain: eFMI® vs. FMI®, AUTOSAR, ASAM, …

Scope of eFMI in embedded software domain

An eFMU is about the *development* of *one* software component (controller, virtual sensor etc) of a complex cyber-physical system:

- Not about system integration of components
  - Many other standards in different industries available (e.g., AUTOSAR, ASAM etc)

  ⇒ Use established standards for eFMU system-integration
- Not about system level programming (embedded OS, drivers, software frameworks etc)

  ⇒ Production Code generators tailor code for given target environment
- Not about distributing, interconnecting and parameterizing system simulations
  - That is what FMI, DCP & SSP are for

  ⇒ Use FMI & co. ecosystem to distribute and setup (desktop environment) system simulations…
  …by exporting your production code as FMU

efmi Functional Mock-up Interface for embedded systems

eFMI vs. FMI: Two complementary standards

FMI: Standardized C interface to enable exchange and interoperability of simulations

- About how to distribute and integrate simulations
- Single abstraction level, 1 ↔ 1 (producer to consumer)
- Focus on interface of black-box implemented functionality

eFMI: Standardized development workspace to implement models in embedded environments

- About how to step-wise develop simulations from high-level model to low-level code
- Chain of abstraction levels, N ↔ M ↔ … ↔ L
  (many development stakeholders with different tools and viewpoints)
- Focus to guarantee non-functional requirements (safety-critical & real-time) besides functional

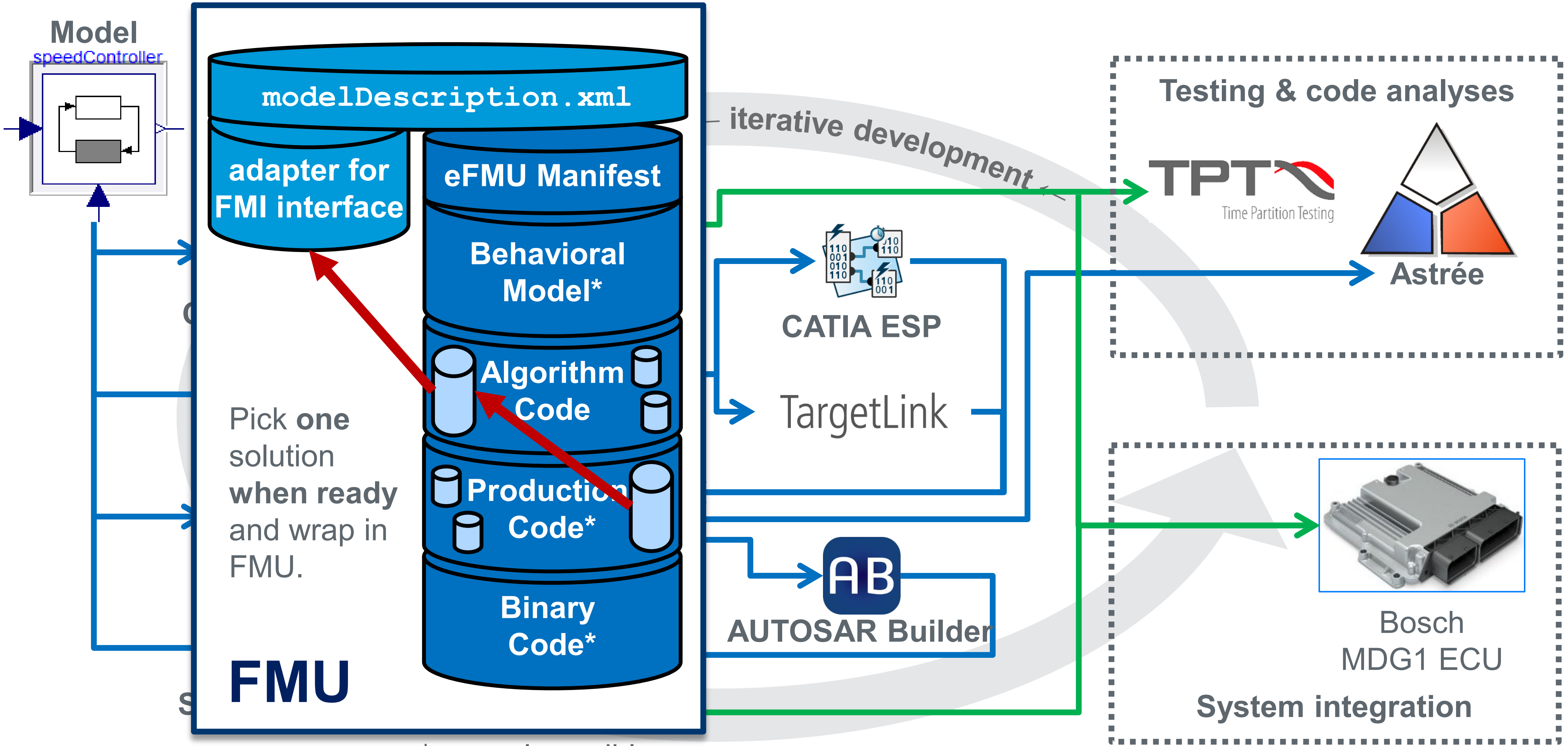
⇒ We can develop functionality with eFMI and distribute it with FMI

⇒ Two complementary standards

efmi Functional Mock-up Interface for embedded systems

Modelica Association

eFMI Standard: Deployment scenarios

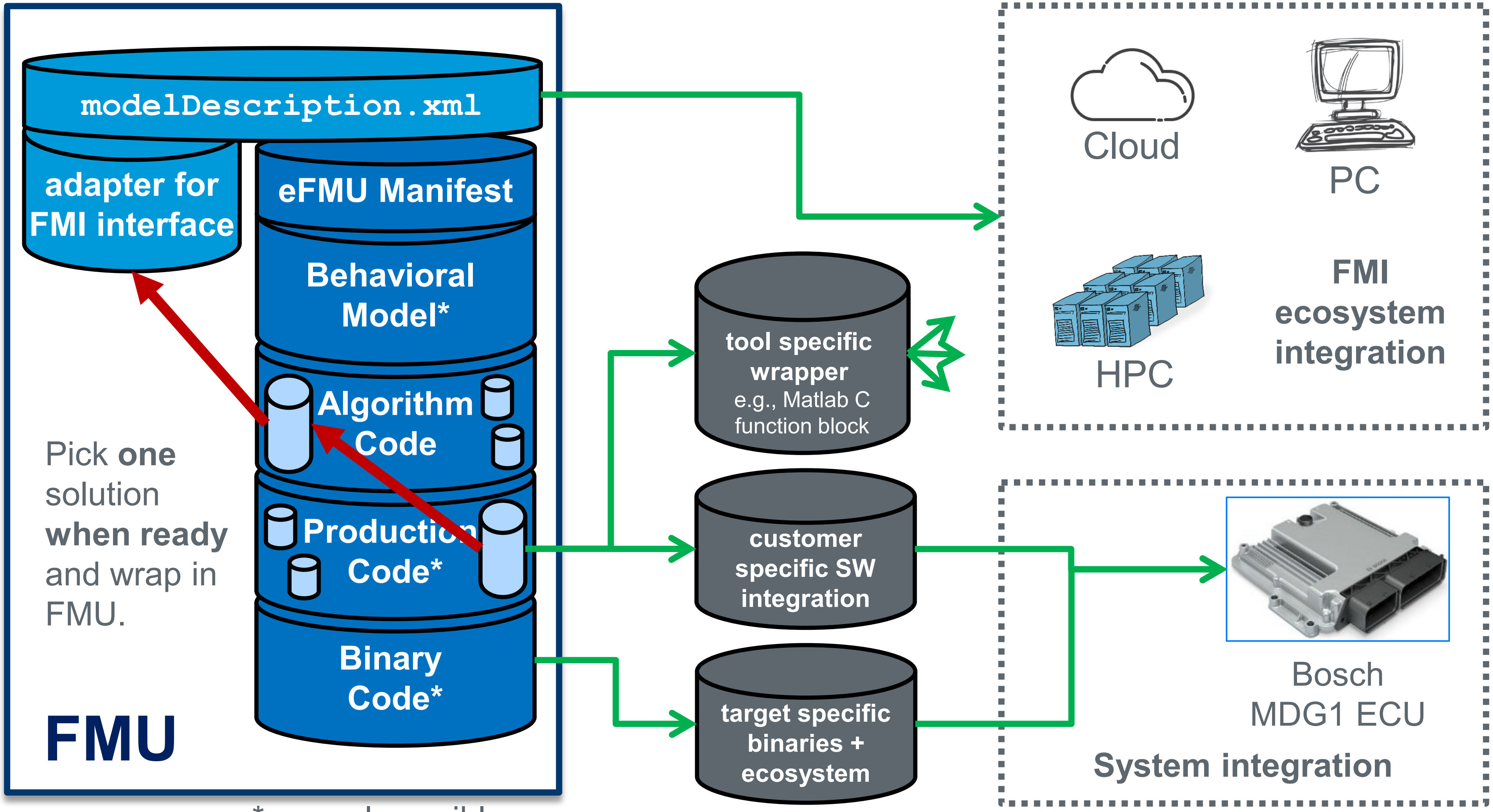


*several possible

eFMI Standard: Deployment scenarios

ASAM
AUTOSAR
MATLAB & SIMULINK

Use existing standards for system integration (**not defined by eFMI**).



*several possible

Modelica Association Project eFMI (MAP eFMI)

efmi Functional Mock-up Interface for embedded systems

Modelon, ALTAIR, DLR, esi, DASSAULT SYSTEMES, dSPACE, BOSCH Invented for life, SIEMENS Ingenuity for life, EFS, iVH Institute of Vehicle Engineering, MITSUBISHI ELECTRIC Changes for the Better, OpenModelica, ETAS, PIKETEC, Mercedes-Benz

Project leader:
Christoff Bürger
DASSAULT SYSTEMES

Deputy project leader:
Hubertus Tummescheit
Modelon

https://efmi-standard.org/