Functional Mock-up Interface

FMI Layered Standard for Network Communication: Applications in Networked ECU Development

16th International Modelica and FMI Conference
Lucerne, Switzerland, Sept 8-10 2025

Christian Bertsch,
Kahramon Jumayev,
Andreas Junghanns,
Pierre R. Mai,
Benedikt Menne,
Masoud Najafi,
Tim Pfitzer
Jan Ribbe,
Klaus Schuch,
Markus Süvern,
Patrick Täuber

Modelica Association

Overview

Motivation and Idea

- Complexity in Automotive Development Cycle
- FMI 3.0: The Foundation for Interoperability

FMI Layered Standard for Network Communication “High Cut”

- Physical Abstraction Layer (“High Cut”)
- Early Architectural Validation with “High-Cut”

FMI Layered Standard for Network Communication “Low Cut”

- Network Abstraction Layer (“Low Cut”)
- Detailed Verification of Software & Network with “Low Cut”

Using FMI Layered Standard for Network Communication across Development Lifecycle

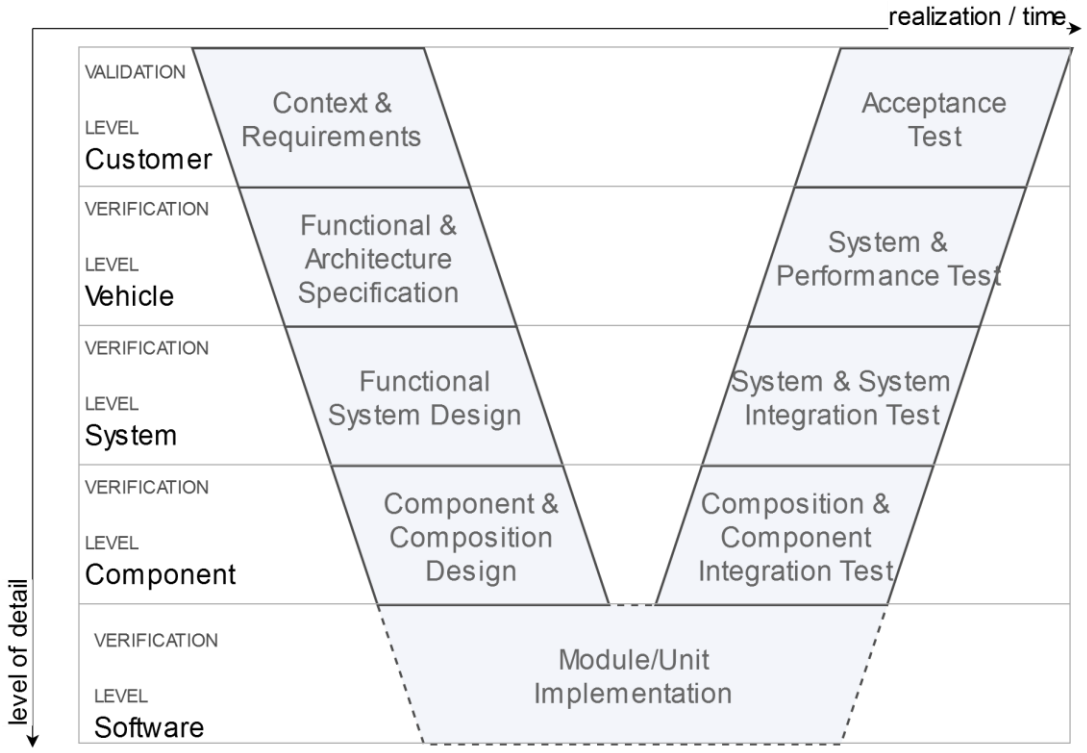
- Bus Simulation Composition with FMI-LS-BUS
- Workflow across the V-Model for Continuous Validation

Take-Away: (R)evolution to SiL Simulation of Virtual ECUs

Complexity in Automotive Development Cycle

The Current Reality: A Disconnected Workflow

- **Increasing Complexity:** Modern vehicles are complex systems-of-systems, with features distributed across numerous ECUs in overall different vehicle architectures.
- **Distributed Development:** Vehicle functions are built by many suppliers using diverse tools, creating integration challenges.
- **Static Specifications:** Collaboration traditionally relies on passing documents, which leads to ambiguity and errors when different teams implement the designs.
- **Late Discovery:** This disconnected workflow means critical integration issues are only found late in the validation phase, resulting in a cycle of costly rework, project delays, and increased risk.



Without a way to test how virtual components communicate early and often, we are forced to find critical, expensive problems too late in the development cycle.

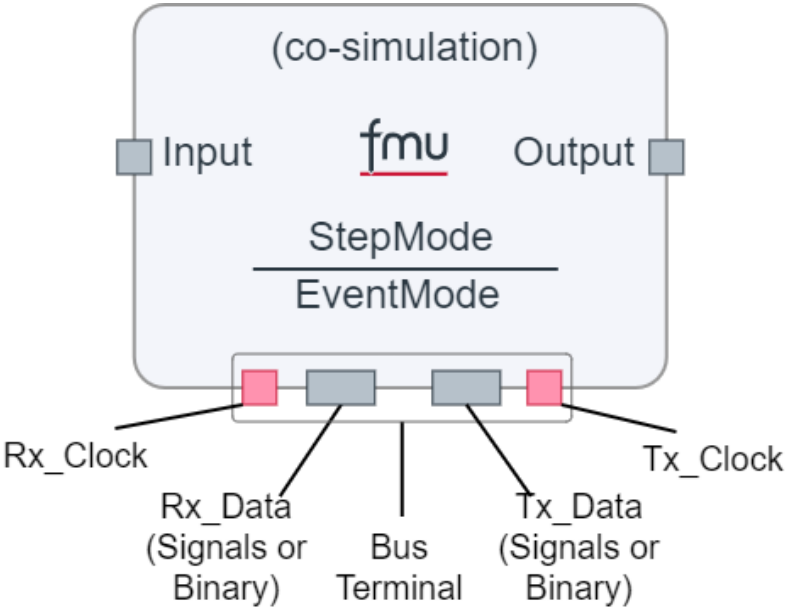
FMI 3.0: The Foundation for Interoperability

Enabling powerful new applications with FMI-LS-BUS:

- **FMI 3.0** is the proven core standard for tool interoperability.
- FMI 3.0's architecture uses **"Layered Standards"** to integrate domain-specific knowledge while keeping the core lean. E.g. FMI project and partners working on topics like **FMI-LS-XCP** and **FMI-LS-STRUCT**
- FMI-LS-BUS support **automotive networks protocols** and others from **different industries** (CAN, LIN, FlexRay, Ethernet, …)
  - **Use FMI 3.0 core standard features** to specify a common bus interface, using **Co-Simulation**, **Signal or Binary Variables**, **Clocks** and **Terminals**

Two variants:

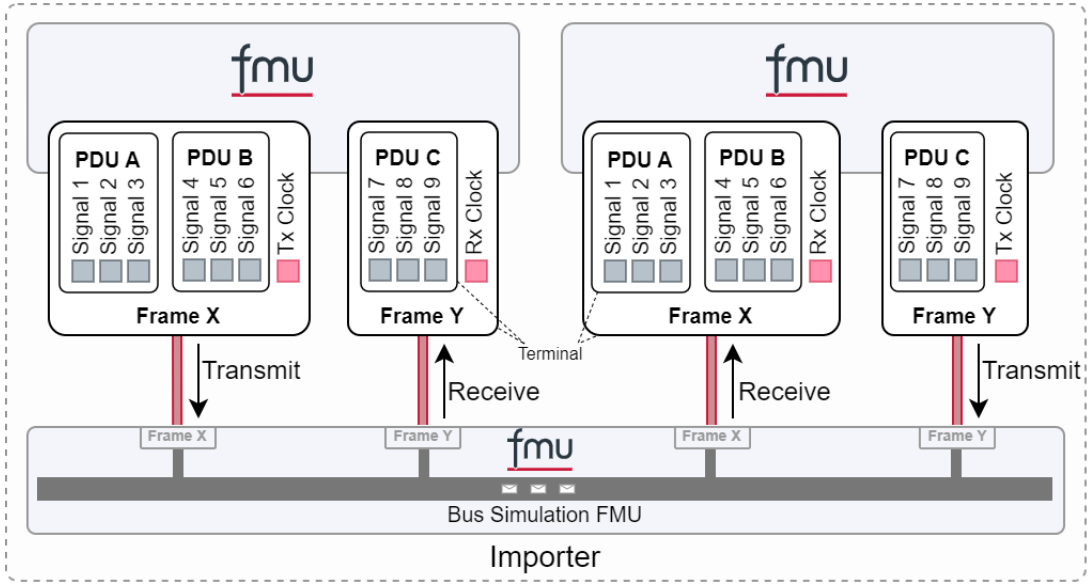
- **Physical abstraction layer ("high cut"):** Simply transport physical signal values between virtual ECUs by using unit-based and clocked variables
- **Network abstraction ("low cut"):** Detailed emulation of a specified bus system to realize virtualized bus driver implementations, including feedback from the physical drivers about transmission status or network node states



Physical Abstraction Layer (“High Cut”)

Technical Breakdown:

- **FMI Variables:** Each bus signal is modeled as a standard fmi variable with its physical type
- **FMI Clocks:** A dedicated clock triggers the data exchange. A tick on a transmission clock (TxClock) indicates that the signal values are new and ready to be sent
- **Hierarchical Terminals:** Terminals are used to group the variables, mirroring the network database structure (Signals -> PDUs -> Frames)

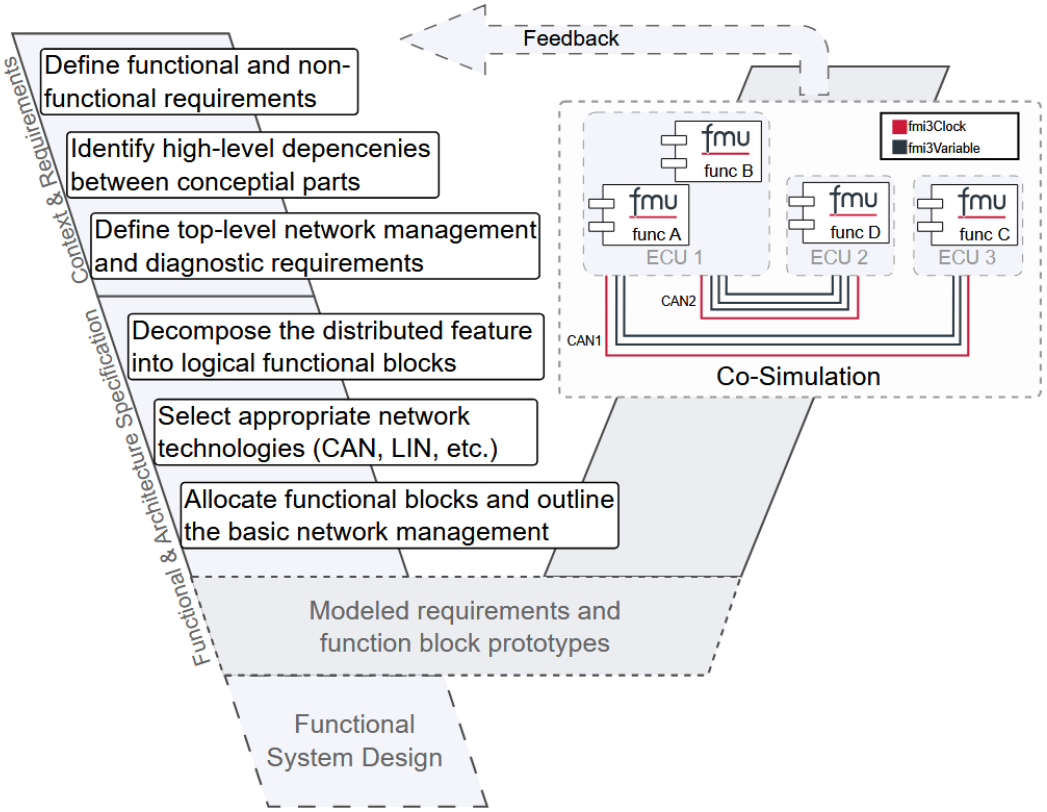



```
<!-- modelDescription.xml -->
<Clock name="Clock" valueReference="1" causality="input" variability="discrete" intervalVariability="constant" intervalDecimal="0.020" shiftDecimal="0.010"/>
<Float64 name="Signal_A" valueReference="3" causality="output" variability="discrete" clocks="1"/>
<Float64 name="Signal_B" valueReference="4" causality="output" variability="discrete" clocks="1"/>

<!-- terminalsAndIcons.xml -->
<Terminal terminalKind="org.fmi-ls-bus.frame-terminal" name="Frame_A" matchingRule="bus">
    <TerminalMemberVariable variableKind="signal" variableName="Clock" memberName="Clock"/>
    <Terminal terminalKind="org.fmi-ls-bus.pdu-terminal" name="PDU_A" matchingRule="bus">
        <TerminalMemberVariable variableKind="signal" variableName="Signal_A" memberName="Signal_A"/>
        <TerminalMemberVariable variableKind="signal" variableName="Signal_B" memberName="Signal_B"/>
    </Terminal>
</Terminal>
```

Early Architectural Validation with “High Cut”

How High-Cut Helps in Early Stages

- **Executable Specifications:** It complements ambiguous static documents with executable FMUs that serve as a testable "single source of truth" for system design.
- **Early Virtual Integration:** Enables teams to test how distributed functions interact in a co-simulation environment long before any hardware is available or even design is fixed.
- **Find Architectural Flaws Sooner:** Allows architects to validate data flows and discover conceptual errors in the design before they propagate into costly, later stages of development.
- **Faster, Cheaper Feedback:** Creates a rapid feedback loop for design and testing, significantly reducing integration risks and costs.

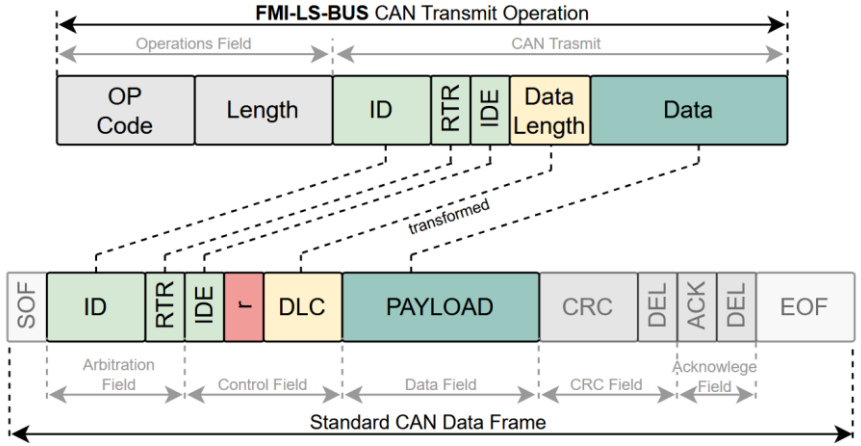


High-Cut provides an executable representation of the system, allowing for early and continuous validation of the architecture's functional behavior.

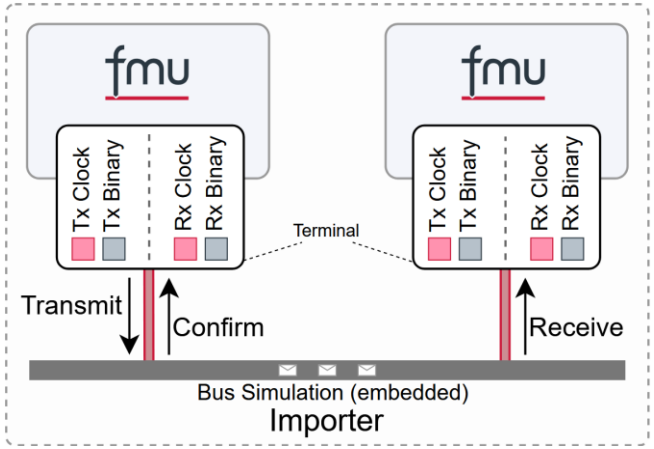
Network Abstraction Layer (“Low Cut”)

Technical Breakdown:

- **Binary Variables:** A single binary variable for each direction (Tx/Rx) acts as the data carrier. It holds a serialized payload a "bus operation" defined by the FMI-LS-BUS standard.
- **FMI Clocks:** A clock is tightly coupled with each binary variable. A tick on the TxClock is the event that signals "the data in TxBinary is valid now and should be processed by the bus simulation".
- **Terminals:** These elements are grouped into a single terminal per bus connection for simple, clean connections in the co-simulation environment.



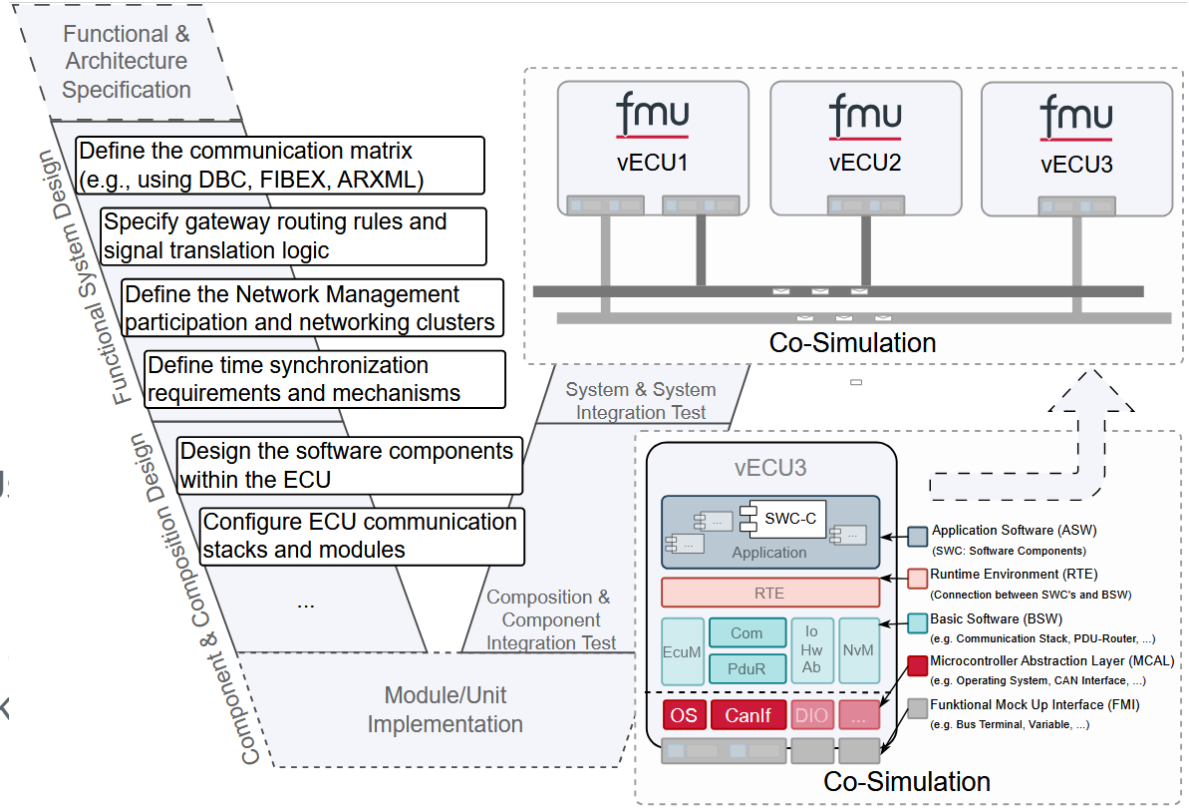

```
<!-- modelDescription.xml -->
<Clock name="Tx_Clock" valueReference="1" causality="input" intervalVariability="countdown" />
<Clock name="Rx_Clock" valueReference="2" causality="input" intervalVariability="triggered" />
<Binary name="Tx_Data" valueReference="3" causality="output" variability="discrete" mimeType="application/org.fmi-standard.fmi-ls-bus.can; version=&quot;1.0.0&quot;" clocks="1" />
<Binary name="Rx_Data" valueReference="4" causality="input" variability="discrete" mimeType="application/org.fmi-standard.fmi-ls-bus.can; version=&quot;1.0.0&quot;" clocks="2" />
<Boolean name="BusNotification" valueReference="5" causality="parameter" variability="fixed" start="false"/>

<!-- terminalsAndIcons.xml -->
<Terminal terminalKind="org.fmi-ls-bus.network-terminal" name="CanController" matchingRule="org.fmi-ls-bus.transceiver">
    <TerminalMemberVariable variableKind="signal" variableName="Tx_Data" memberName="Tx_Data" />
    <TerminalMemberVariable variableKind="signal" variableName="Tx_Clock" memberName="Tx_Clock" />
    <TerminalMemberVariable variableKind="signal" variableName="Rx_Data" memberName="Rx_Data" />
    <TerminalMemberVariable variableKind="signal" variableName="Rx_Clock" memberName="Rx_Clock" />
    <TerminalMemberVariable variableKind="signal" variableName="BusNotification" memberName="BusNotification" />
</Terminal>
```



Detailed Verification of Software & Network with “Low Cut”

How Low-Cut Helps in Later Stages:

- **Verify Communication Software:** Allows developers to test the vECU's software stack, catching critical bugs like incorrect signal packing or protocol deviations before hardware integration.
- **Realistic Network Analysis:** Enables comprehensive testing of network behavior, including bus load, message latency, and timing effects.
- **Full System Integration:** Test teams can combine vECU from multiple suppliers into a complete virtual system to validate end-to-end performance.
- **Test for Robustness:** Allows for the systematic injection network errors to ensure error-handling mechanisms work as required

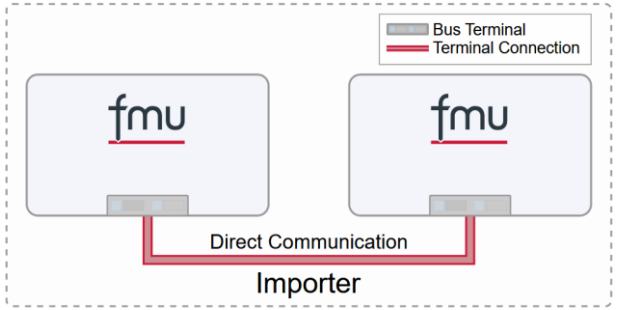


Low-Cut enables the detailed verification of the actual communication software and network behavior, ensuring the final system is robust and correct.

Bus Simulation Composition with FMI-LS-BUS

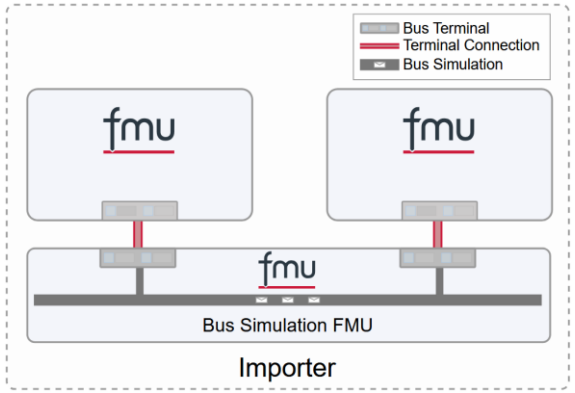
One Interface, Maximum Flexibility

- The same standardized FMU can be used across all three compositions—from simple direct connections to complex, realistic bus simulations - without requiring any changes to the FMU itself



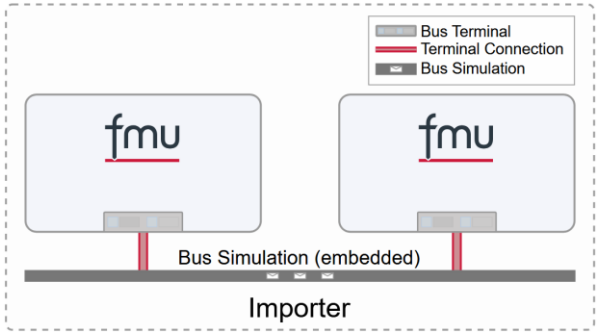
Bus simulation idealized

- 1:1 connection
- Idealized bus
  (only ideal timing, no bandwidth limits, …)
- Standard FMI Importer can be used



Bus simulation via Bus Simulation FMU

- n:m connection
- Bus behavior can be simulated
- Standard FMI Importer can be used
- Can be generated based on network descriptions
- Changes of on system level require regeneration



Bus simulation supported by the Importer

- n:m connections
- Bus behavior can be simulated
- Dynamic configuration
- No network descriptions needed

Workflow across the V-Model for Continuous Validation

Phase 1: Architecture Validation (High-Cut):

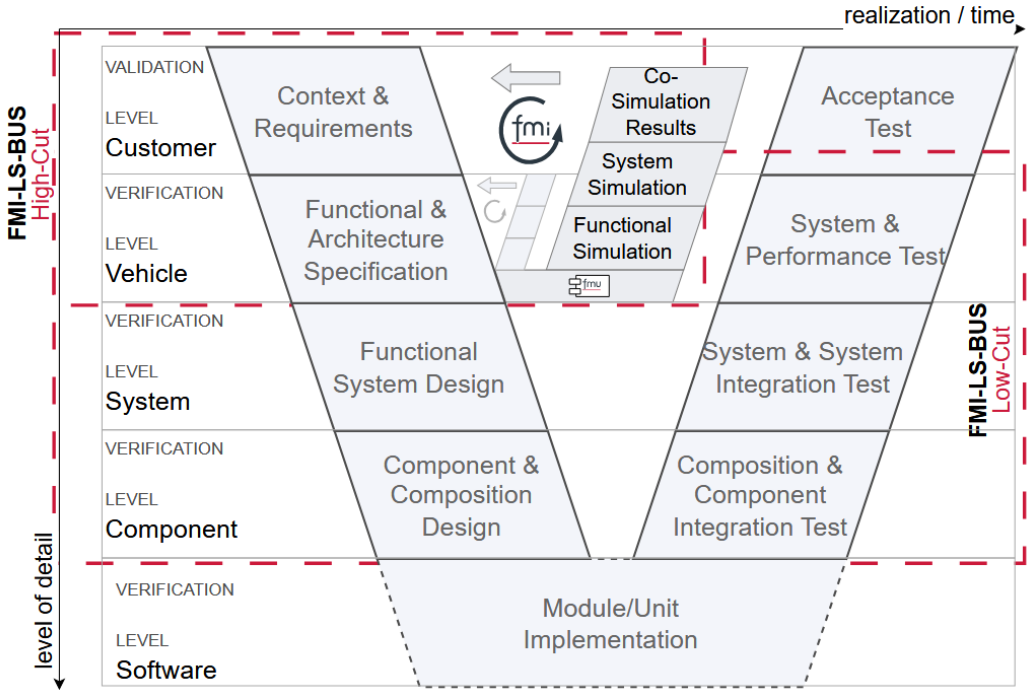
- Early development phases use **High-Cut** to model and test the functional behavior at a signal level.
- This establishes a validated, executable architecture before detailed implementation begins, replacing ambiguous documents with a clear, functional model.

Phase 2: Implementation & Verification (Low-Cut):

- As development progresses, the validated architecture is refined into detaile software components and virtual ECUs.
- **Low-Cut** is then used to verify the actual communication software, network protocols, and end-to-end system timing at the message level.

The Benefit: A Unified Process

- This integrated approach enables a modern, iterative workflow that supports Continuous Integration and Continuous Testing (CI/CT) practices.
- It closes the gap between design, implementation, and testing, which significantly reduces integration risks and enhances overall efficiency.



By combining High-Cut for early architecture and Low-Cut for detailed software verification, FMI-LS-BUS enables a continuous, reliable testing process across the entire V-model.

Take-Away: (R)evolution to SiL Simulation of Virtual ECUs

FMI-LS-BUS is a key enabler for modern virtual automotive development

- The FMI standard alone is available for exchanging simple virtual ECUs in early development phases
- Standardizes network simulation as an extension to FMI 3.0 with FMI-LS-BUS
- Flexible High-Cut and Low-Cut layers align perfectly with the development lifecycle
- Enables early, continuous validation and improves collaboration and tool agnostics

The advantages for validation are obvious:

- **Lower integration costs** thanks to **better interoperability** of tools and test platforms from **different vendors** within virtual validation scenarios
- More **efficient collaboration** thanks to **significantly reduced coordination effort** between **OEMs and suppliers**

Outlook: FMI Layered Standard for Network Communication

Released Version FMI-LS-BUS v1.0.0

- FMI Layered Standard for Network Communication 1.0 released - enabling the simulation of virtual ECUs with FMI 3.0!
- **More information:**
  - FMI Layered Standard for Network Communication (FMI-LS-BUS) v1.0.0, (Github repository and Issue tracker)

Tool Support and Contribution:

- The adoption of the industry including implementations, prototypes and tool releases supporting the FMI Layered Standard for Network Communication is rapidly growing.

  Many thanks to all contributors!

Listed Tools for FMI-LS-BUS:

- Altair (Twin Activate)
- AVL (Model.CONNECT)
- Akkodis (PROVEtech:RE)
- Bosch (Power Solutions)
- dSPACE (SystemDesk, VEOS)
- Synopsys (Silver)
- Vector (SIL Kit FMU Importer, vVIRTUALtarget)

Questions?

Comments?

Ideas?

Feedback?

Join the FMI LinkedIn Group!