Published online · July 2026

Michał Kleiber
Eugenio Oñate
*Editors*

# Bridging the Embedded Execution Gap: A Review of the Functional Mock-up Interface for Actuator-Centric Execution Workflows in Industrial Embedded Systems

Archives of Computational Methods in Engineering

State of the Art Reviews

CIMNE
BARCELONA

Springer

Archives of Computational Methods in Engineering
https://doi.org/10.1007/s11831-026-10715-3

REVIEW

# Bridging the Embedded Execution Gap: A Review of the Functional Mock-up Interface for Actuator-Centric Execution Workflows in Industrial Embedded Systems

**Sebastian Rojas-Ordoñez[1,2] · Mikel Segura[1] · Irune Yarza[1] · Ekaitz Zulueta[2]**

Received: 30 March 2026 / Revised: 3 June 2026 / Accepted: 28 June 2026
© The Author(s) 2026

**Abstract**

Industrial actuators integrate multi-domain physical dynamics with embedded control algorithms under strict timing constraints. Despite mature model-based design workflows, a persistent embedded execution gap separates validated high-fidelity simulation models from deterministic, resource-bounded execution on embedded industrial platforms. This review examines the Functional Mock-up Interface (FMI) as an interoperability enabler for actuator development workflows and argues that interoperability at the model-exchange level does not automatically translate into interoperability at the execution level. The review analyzes the evolution of FMI from version 2.0 to 3.0, systematizes actuator-relevant fidelity requirements (nonlinearities, compliance/resonance, timing effects), and evaluates representative approaches for integrating FMI into industrial and robotic toolchains. Two deployment trajectories are synthesized: (i) runtime-constrained FMU execution on edge/MCU-class platforms for prototyping, monitoring, and non-safety-critical functions; and (ii) analyzable code-generation paths such as Embedded FMI (eFMI) and GALEC, which target bounded memory and timing transparency for certification-oriented workflows. Although FMI 3.0 introduces improved timing semantics through clocks and Scheduled Execution, end-to-end feasibility remains limited by toolchain maturity, scheduler integration patterns, and the scarcity of complete quantitative evidence on jitter, WCET, and memory footprint under realistic load. The review identifies key challenges in determinism, fidelity management, certification alignment, and ecosystem-level conformance/validation, clarifying the technical barriers that must be addressed to fully bridge the gap between high-fidelity simulation and embedded actuator deployment.

## 1 Introduction

Industrial applications increasingly depend on complex mechatronic systems in which actuators play a pivotal role in meeting performance, safety, and reliability requirements. In domains such as industrial robotics, manufacturing automation, and cyber-physical production systems, actuators have evolved from passive components into software-defined electromechanical subsystems. These units combine physical dynamics, power electronics, control algorithms, and embedded software into tightly coupled components [1–3].

As a result, the design and validation of actuator control systems require high-fidelity models that capture nonlinearities, compliance, friction, time delays, and embedded execution constraints. Beyond classic control-theoretic requirements, fidelity in actuators is often dominated by contact/friction phenomena and compliance-related

✉ Sebastian Rojas-Ordoñez
jsrojas@ikerlan.es; jrojas015@ikasle.ehu.eus

Mikel Segura
msegura@ikerlan.es

Irune Yarza
iyarza@ikerlan.es

Ekaitz Zulueta
ekaitz.zulueta@ehu.eus

[1] IKERLAN Technology Research Centre, Paseo José María Arizmendiarrieta 2, Arrasate, Gipuskoa 20500, Spain

[2] Systems Engineering and Automation, University of the Basque Country (UPV/EHU), Nieves Cano 12, Vitoria-Gasteiz, Álava 01006, Spain

Published online: 15 July 2026

resonances, which can strongly affect stability margins and achievable bandwidth [4–8]. Crucially, actuator-centric models are frequently stiff and event-rich, so their predictive value depends not only on physical realism but also on numerical integration choices, event handling, and time-step semantics. Therefore, high-fidelity models must support reproducible verification, code generation, and integration into hardware-in-the-loop (HIL) and embedded execution contexts, where timing and resource constraints become first-class design variables [9, 10]. In practice, this means that actuator fidelity is inseparable from the computational contract under which the model executes: solver availability, tolerance policies, event localization rules, and the communication-step strategy in co-simulation directly shape the closed-loop response.

Despite progress in model-based systems engineering, industrial development remains fragmented. High-fidelity actuator models are often created in specialized simulation tools (e.g., MATLAB/Simulink/Simscape, Dymola, Amesim, MapleSim), whereas control logic is developed separately using embedded toolchains or PLC platforms. This toolchain fragmentation impedes model reuse and interoperability across the development lifecycle, forcing engineers to reimplement or simplify deployment models [11–13]. These gaps compromise traceability and introduce errors due to semantic mismatches between modeling environments and embedded targets, a challenge repeatedly reported in digital engineering and digital twin adoption studies [14, 15]. From a computational standpoint, fragmentation also prevents systematic regression testing: the same model may be executed under different solvers, different stepping policies, and different scheduling assumptions across phases, making discrepancies hard to diagnose and hard to certify.

This fragmentation gives rise to a critical but insufficiently addressed problem: the embedded execution gap. We define the embedded execution gap as the structural mismatch between:

1. Validated high-fidelity simulation artifacts.
2. Their deterministic, resource-bounded, and certifiable execution on embedded targets.

In particular, validated behavior in a desktop simulation setting does not guarantee equivalent behavior under embedded execution semantics, where solver availability, scheduling policies, memory budgets, and certification-driven analyzability constraints can change the closed-loop response. This gap is not merely an engineering inconvenience: in actuator loops, small timing deviations (jitter), solver-induced phase lag, or event-handling artifacts can propagate into stability margins, constraint violations, and safety envelopes.

While high-fidelity actuator models can be validated in simulation environments, their direct deployment onto deterministic embedded platforms remains non-trivial. Differences in numerical solvers, scheduling semantics, memory constraints, and certification requirements often prevent validated models from being reused without substantial redesign. As a result, interoperability at the modeling level does not automatically translate into interoperability at the execution level. This distinction is especially critical for actuator loops operating at high bandwidth, where microsecond-to-millisecond timing deviations and numerical artifacts can propagate into stability and safety margins. Consequently, interoperability standards that focus on syntactic exchange or tool connectivity may remain insufficient unless temporal semantics and execution guarantees are made explicit. In co-simulation, this execution-level dimension is largely governed by the master algorithm (communication step size, extrapolation strategy, and roll-back/iteration support), while in embedded deployment it is governed by RTOS task models, scheduling policies, and bounded-memory execution.

Figure 1 analytically frames the embedded execution gap that motivates this review and previews the two main deployment trajectories discussed throughout the paper. The figure distinguishes interoperability at the model level, enabled by FMI-based FMU exchange, from interoperability at the execution level, which is constrained by real-time scheduling, bounded-memory requirements, and certification-driven analyzability. Critically, the figure makes explicit that the gap is not a single boundary but a layered set of constraints (execution semantics, timing contracts, and certification evidence) that each require specific engineering responses, mapped to the deployment archetypes (A1–A4) formally defined in Sect. 7.

The embedded execution gap emerges at the transition from validated simulation artifacts to production-grade embedded control. Two mitigation trajectories are highlighted:

1. Runtime-constrained FMU adaptation, which partially mitigates execution mismatches through restricted stepping and limited scheduling guarantees
2. eFMI/GALEC-based deterministic code generation, which structurally reduces the gap by enforcing analyzable timing and bounded-memory properties suitable for certification-oriented workflows.

These two families are reviewed in Sects. 7 and 8. Importantly, these trajectories differ not only in deployment mechanics but also in what can be proven. Runtime reuse tends to provide feasibility evidence (often

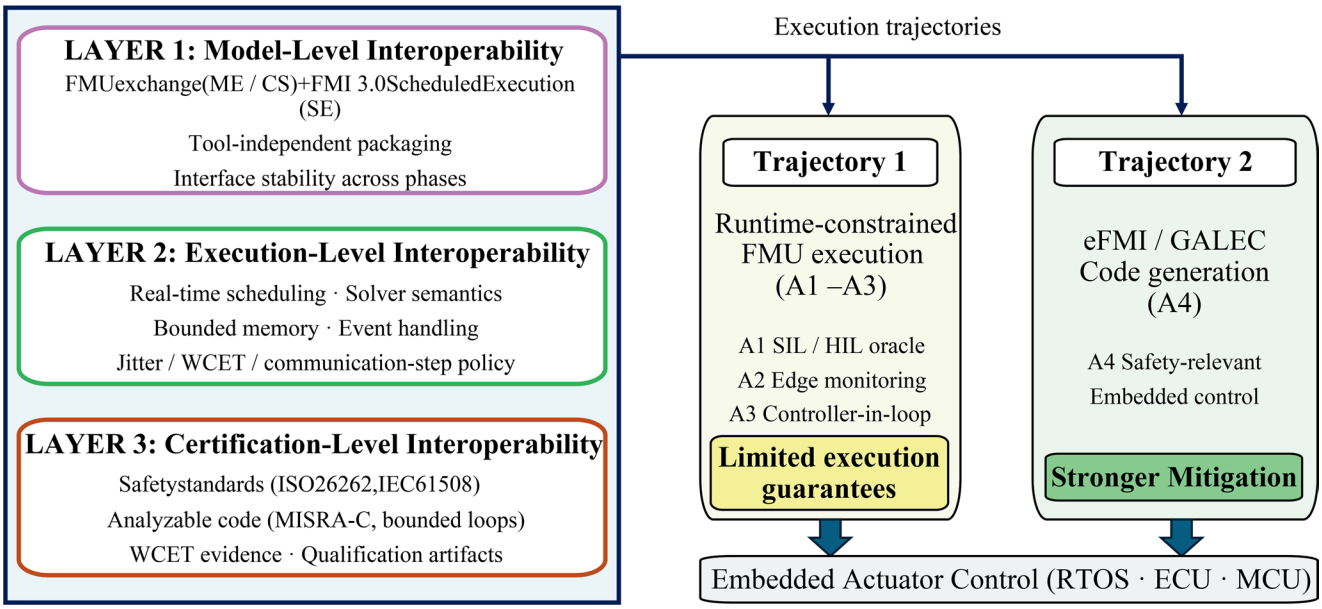



**Fig. 1** Analytical framework of the embedded execution gap in actuator-centric industrial systems. Three interoperability layers are distinguished: **(i)** model-level portability, enabled by FMU packaging and exchange (FMI 2.0/3.0); (ii) execution-level constraints, driven by real-time scheduling, bounded-memory policies, solver semantics, and event-handling overhead; and (iii) certification-level requirements, governed by WCET analyzability, safety-standard alignment, and qualification evidence. Two mitigation trajectories are shown: runtime-constrained FMU adaptation (trajectory 1, partial determinism, suitable for deployment archetypes A1–A3 defined in Sect. 7) and eFMI/GALEC-based deterministic code generation (trajectory 2, certification-oriented, targeting archetype A4). The gap regions indicate where interoperability at the model level does not automatically translate to interoperability at the execution or certification level

platform-dependent), whereas analyzable code-generation aims to provide timing/memory arguments compatible with safety cases.

The challenge is further amplified in embedded and edge computing applications. Modern actuators increasingly integrate local computation to support real-time control, diagnostics, and adaptation. However, embedded platforms impose strict limitations on memory footprint, compute capacity, and execution determinism [16, 17]. Safety standards such as ISO 26262 or IEC 61508 further demand predictable Worst-Case Execution Times (WCET), which complicates the direct deployment of complex, black-box simulation models [18, 19]. This motivates a computational view of interoperability: models must be portable not only across tools but also across execution semantics, with verifiable timing and resource bounds. Accordingly, claims of embedded readiness must be grounded in explicit activation assumptions (sampling period and deadline), timing evidence (WCET/jitter under load), and bounded-memory policies, rather than solely in FMU exchange success.

To bridge this gap, standardized interfaces must decouple model semantics from execution backends while preserving numerical consistency, temporal determinism, and certification-relevant properties [9, 10]. The Functional Mock-up Interface (FMI) has become a widely adopted standard for tool-independent model exchange and co-simulation [11, 20]. FMI encapsulates simulation models as Functional Mock-up Units (FMUs), enabling cross-tool reuse and system integration. However, co-simulation introduces well-known challenges in synchronization, numerical stability, and determinism that become particularly critical for high-bandwidth actuator loops [21, 22]. While FMI has seen broad uptake in desktop co-simulation and system-level modeling, its applicability to embedded control and actuator-centric applications is still evolving. FMI 3.0 clocks and Scheduled Execution address timing semantics at the interface level [23, 24]. Embedded FMI and GALEC-based code generation address a separate deployment path oriented toward analyzable embedded artifacts [25–27]. In addition, the computational framing of FMI 3.0 (clocked variables, partitioned execution, and scheduler-controlled activation) creates a natural interface to classical real-time analysis and to certification-oriented constraints, but only if toolchains expose and validate the corresponding timing and resource contracts.

This review investigates how FMI standards enable interoperability for high-fidelity simulation and actuator control in embedded industrial systems. Unlike existing surveys that primarily emphasize generic digital twin architectures or system-level co-simulation [21], this work positions the embedded execution gap as the defining interoperability challenge for actuator-centric industrial systems. Accordingly, the review emphasizes computational execution semantics (scheduling, step coordination, and determinacy)

as the bridge between modeling interoperability and deployable embedded control. It analyzes the evolution of the FMI specification, reviews strategies for deploying stiff actuator models, and assesses representative tools that integrate FMI into robotic/industrial workflows [28–31]. To strengthen the state of the art, the review also consolidates recurring execution failure modes (e.g., coupling-induced oscillations, host-dependent trajectories, and latency inflation in closed-loop settings) and maps them to practical mitigation strategies and reporting checklists. The main contributions of this paper are as follows:

- A structured analysis of FMI 2.0 versus FMI 3.0 with respect to execution semantics, focusing on clocks, scheduled execution, and their implications for real-time analyzability on embedded platforms.
- A comparative review of deployment strategies for high-fidelity actuator models under resource and timing constraints, distinguishing between runtime interpretation and deterministic code generation via eFMI/GALEC.
- A critical assessment of the FMI tooling ecosystem from a computational perspective, identifying gaps in determinism, numerical robustness, certification readiness, and support for actuator nonlinearities.
- A computational synthesis layer that distinguishes model-level interoperability from execution-level interoperability and proposes execution-contract reporting elements (timing, memory, and solver/stepping disclosure) to improve reproducibility and industrial credibility.

The remainder of this paper is organized as follows. Section 2 describes the review methodology. Section 3 introduces the FMI standard's evolution. Section 4 discusses specific actuator modeling requirements. Section 5 analyzes interoperability across the control stack. Section 6 consolidates computational determinism mechanisms (master algorithms and real-time execution contracts). Section 7 focuses on execution in embedded/edge platforms. Section 8 surveys the tooling ecosystem. Section 9 outlines research gaps, and Sect. 10 concludes the paper.

## 2 Review Methodology

This article follows a structured literature review methodology to analyze the role of the Functional Mock-up Interface (FMI) as an interoperability standard for high-fidelity simulation and actuator-centric control in industrial embedded systems. The review balances amplitude and depth by combining:

1. Normative standards and specification documents defining FMI semantics.
2. Peer-reviewed research and industrial case studies addressing embedded execution, real-time constraints, and actuator-relevant applications.

To avoid conflating normative sources with empirical evidence, the review explicitly separates (a) normative and implementer sources (standards, implementers' guides, and tooling documentation) from (b) the peer-reviewed research corpus used for PRISMA-based synthesis.

To strengthen transparency and reproducibility, the methodology is reported using a *PRISMA-lite* structure adapted for engineering reviews [32]. The PRISMA-guided selection process applies exclusively to peer-reviewed research articles addressing actuator-centric FMI usage. Standards, industrial guidelines, and tool documentation are included as contextual references but are treated distinctly from the research corpus. Accordingly, all PRISMA counts, exclusion codes, and quality scores refer only to the peer-reviewed set, whereas normative documents are tracked in a separate reference list supporting specification-level interpretation.

### 2.1 Scope and Research Questions

The scope targets FMI-based workflows that directly affect the feasibility of deploying high-fidelity models in embedded actuator-control contexts. The analysis covers:

1. FMI 2.0 and FMI 3.0 execution modes (Model Exchange, Co-Simulation, and Scheduled Execution).
2. Timing semantics, specifically clocked systems and scheduler integration.
3. Deployment targets, including Edge computing, Microcontrollers (MCU), and Hardware-in-the-Loop (HIL) setups.
4. Emerging embedded-oriented standardization, specifically the Embedded Functional Mock-up Interface (eFMI) and analyzable code generation workflows.

The scope is intentionally actuator-centric and execution-centric: papers are included only if they provide sufficient technical detail to reason about numerical stepping and/or timing semantics (e.g., synchronization policies, scheduling assumptions, deadlines, jitter/WCET, or bounded-resource execution). Studies that mention FMI only as a packaging artifact (without execution assumptions) are treated as contextual but excluded from PRISMA synthesis.

The review is guided by the following research questions:

- **RQ1:** Which FMI concepts and execution modes are most relevant to actuator-centric embedded control, and what technical limitations remain?
- **RQ2:** What evidence exists regarding the real-time feasibility of FMI-based execution on embedded and edge platforms?
- **RQ3:** Which engineering patterns and toolchains enable the transition from high-fidelity FMUs to embedded deployment?
- **RQ4:** What interoperability, determinism, and certification-related challenges limit plug-and-play adoption of FMI for industrial actuators?

## 2.2 Literature Search Strategy

The literature search was conducted across Web of Science, Scopus, IEEE Xplore, SpringerLink, MDPI, and Google Scholar. Web of Science was included given its strong coverage of high-impact engineering journals, particularly in embedded systems, real-time control, and mechatronics, domains central to this review. In addition, authoritative technical reports, standard specifications, implementers' guides, and tool documentation published by the Modelica Association and industrial stakeholders were included to capture both specification-level evolution and implementation practice [20, 23, 33].

The search covered publications from 2015 to early 2026. The start date was selected to capture the consolidation and maturity phase of FMI 2.0, while the post-2020 emphasis targets FMI 3.0 developments and embedded execution initiatives. Searches were restricted to English-language records. Journal articles and peer-reviewed conference proceedings were prioritized for the PRISMA synthesis; standards and implementers' documentation were included irrespective of venue, given their normative role.

Keyword combinations covered FMI fundamentals, scheduling semantics, and embedded actuator-control contexts. Representative Boolean queries included:

- `("Functional Mock-up Interface" OR FMI) AND (embedded OR MCU OR RTOS OR "real-time")`
- `(FMI OR FMU) AND (determinism OR WCET OR jitter OR scheduling)`
- `(FMI OR "Scheduled Execution") AND ("actuator control" OR drive OR HIL)`
- `(eFMI OR GALEC) AND (embedded OR AUTOSAR)`

**Table 1** Search protocol summary

| Item | Description / Value |
|---|---|
| Databases | Web of Science; Scopus; IEEE Xplore; SpringerLink; MDPI; Google Scholar. |
| Time Window | 2015–2026 (Emphasis on post-2020 for FMI 3.0/eFMI). |
| Core Themes | FMI/FMU + Embedded/Real-Time + Actuator/Drives + eFMI/GALEC. |
| Screening | Title/Abstract → Full-text Eligibility. |
| Exclusions | Standards and tool documentation excluded from PRISMA counts (treated as normative references). |

To reduce database-indexing bias (particularly for recent FMI 3.0 and eFMI work), the search was complemented with:

1. Targeted queries on Scheduled Execution and clocks.
2. Backward/forward snowballing from key standards and highly cited FMI surveys.
3. A supplementary Web of Science (WoS) search conducted in response to reviewer feedback, using the topic query `("Functional Mock-up Interface" OR "FMI" OR "FMU") AND ("actuator" OR "embedded" OR "real-time" OR "drive" OR "robot")` with no time restriction and English-language filter. This returned 25 records; after removing 1 duplicate already in the corpus (WOS:000881031200001, same DOI as S1-038) and applying the same screening criteria, 8 new studies were added to S1-A, 4 to S1-B (contextual), and 12 were excluded under code E2 (non-actuator-centric domain).

Table 1 summarizes the search protocol for peer-reviewed studies.

## 2.3 Study Selection, Duplication Removal, and Screening

All retrieved records were consolidated into a single reference corpus, with duplicate removal using automated matching based on title, authors, year, and DOI, followed by manual verification for ambiguous cases (e.g., conference papers later published as journal articles). When both conference and journal versions existed, the journal version was retained as the primary reference, unless the conference version contained unique implementation details not preserved in the journal extension (in which case both were retained but counted once for PRISMA). Such dual-version cases are explicitly flagged in Supplementary S1-A using a dual-entry annotation, identifying which version is treated as primary for quality scoring and synthesis.

Regarding source types: records accessed via preprint servers (arXiv) or repository platforms (ResearchGate) were included in the PRISMA-lite corpus only when (i) no peer-reviewed counterpart could be identified at the time of the search, and (ii) the record provided unique tooling, implementation, or experimental details not available elsewhere

in the indexed literature. Such entries are explicitly flagged in S1-A and excluded from quality-scored synthesis. All other preprints or repository records are treated as contextual references in the Extended Evidence Inventory (S1-B).

Studies addressing related but non-actuator-centric domains (e.g., building energy systems, district heating networks, or generic Modelica regression testing infrastructure) were not included in the PRISMA-lite synthesis corpus (S1-A). Inclusion in S1-A required at least one of the following: explicit treatment of actuator or drive dynamics; embedded or real-time execution constraints relevant to actuator-centric control; or direct application to industrial mechatronic, robotic, or cyber-physical systems involving actuation. Domain-adjacent studies that provide useful methodological context without meeting these criteria are included exclusively in S1-B as contextual references.

As shown in Table 2, screening followed a two-stage process:

1. Title and abstract screening to remove clearly out-of-scope records.
2. Full-text assessment to confirm FMI relevance and sufficient technical depth for actuator-centric embedded control.

Full-text exclusions were logged with explicit reason codes to ensure auditability.

**Table 2** PRISMA-lite summary of the literature selection process (peer-reviewed corpus only)

| Stage | Count (n) |
|---|---|
| Records identified (research databases) | 742 |
| Records additionally identified (Web of Science supplementary search) | 25 |
| Records after duplicate removal | 542 |
| Records screened (title/abstract) | 542 |
| Records excluded (title/abstract) | 381 |
| Full-text assessed for eligibility | 161 |
| Full-text excluded (insufficient technical relevance or non-actuator-centric domain) | 122 |
| Studies included in qualitative synthesis | 36 |

Note: Normative sources (standards, implementers' guides, and tooling documentation) were tracked separately to support specification-level interpretation and ecosystem context, and are explicitly excluded from PRISMA counts

Figure 2 visually summarizes the identification, screening, eligibility, and inclusion stages of the PRISMA-lite workflow applied to peer-reviewed studies. The diagram complements Table 2 by providing a structured overview of record flow and exclusion logic.

Inclusion and exclusion criteria were defined a priori and applied consistently during screening. Publications were included if they:

- Explicitly addressed FMI (v2.0 and/or v3.0) or embedded-focused variants such as eFMI.

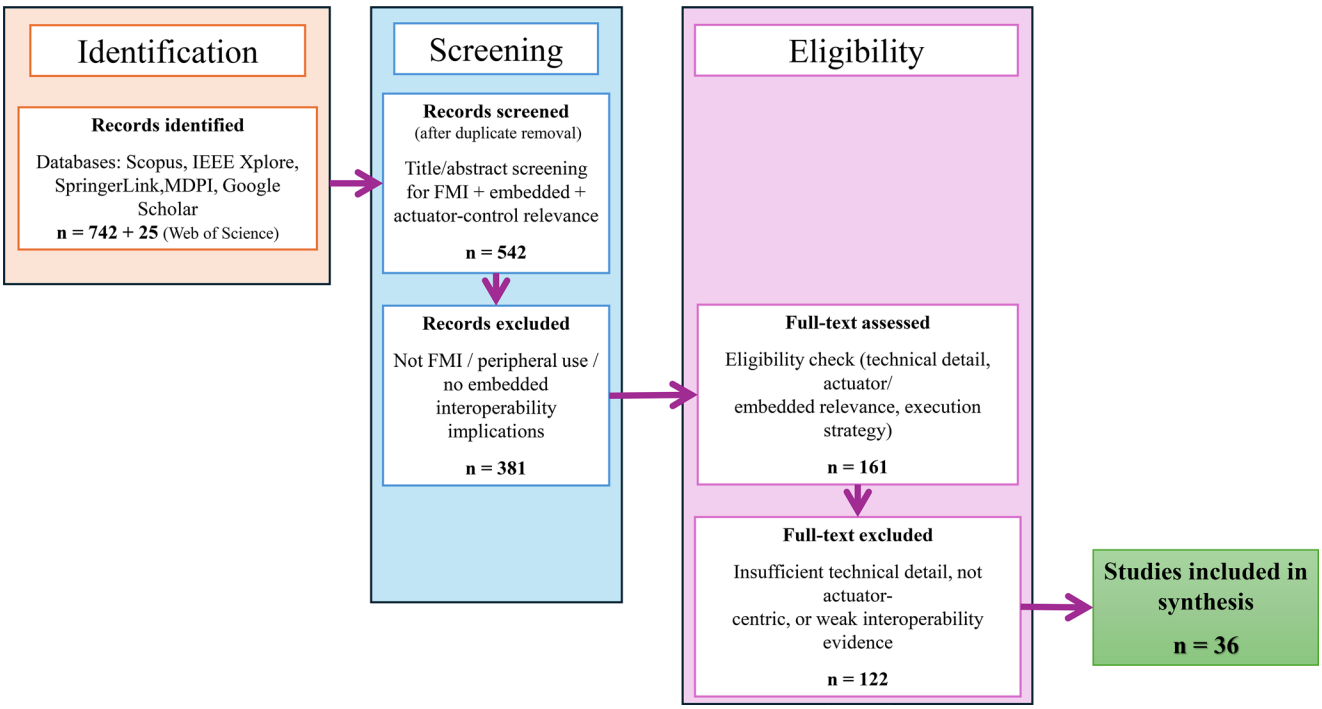



**Fig. 2** PRISMA-lite flow diagram describing the study selection process for the peer-reviewed corpus. Normative sources (standards, implementers' guides, and tooling documentation) were tracked separately and are excluded from PRISMA counts

**Table 3** Full-text exclusion reason codes used in the PRISMA-lite screening ($n=122$)

| Code | Reason for exclusion at full-text stage | Count |
|---|---|---|
| E1 | FMI mentioned only superficially (e.g., FMU used as a black-box artifact) without technical integration details relevant to actuator modeling or control. | 18 |
| E2 | Not actuator-centric: focus on generic digital twins, system orchestration, generic Modelica tooling, building/energy systems, or IT/PLM integration without explicit actuator, drive, or control-loop dynamics. Count updated in revision to include 22 studies retroactively reclassified from the initial corpus, plus 12 additional records from the Web of Science supplementary search, after stricter application of actuator-centric inclusion criterion. | 55 |
| E3 | No embedded or edge execution implications: desktop-only workflows, with no discussion of MCU/RTOS deployment, memory constraints, or execution on embedded targets. | 19 |
| E4 | Insufficient real-time or determinism evidence: lack of discussion on deadlines, scheduling, synchronization, WCET, jitter, or closed-loop timing behavior. | 15 |
| E5 | Insufficient technical depth or reproducibility: missing model, toolchain, or configuration details required for taxonomy classification or comparative analysis. | 9 |
| E6 | Duplicate or extended publication: earlier or partial version excluded in favor of a more complete peer-reviewed record. One additional duplicate (S1-312, identical DOI to S1-039) identified during initial revision; one further duplicate identified in Web of Science supplementary search (WOS:000881031200001, same DOI as S1-038). | 6 |
| **Total** | | **122** |

**Table 4** Quality assessment summary of the studies included in the final PRISMA synthesis

| Assessment Criterion | Score Definition (0–2) | Median Score | Observed Trend |
|---|---|---|---|
| Technical specificity | 0: Conceptual only; 1: Partial technical detail; 2: Full technical specification | 1 | Most studies describe FMI workflows but omit low-level execution details. |
| Embedded execution evidence | 0: No embedded context; 1: Simulated embedded assumptions; 2: Real embedded execution | 1 | Embedded feasibility often inferred rather than experimentally demonstrated. |
| Timing and determinism awareness | 0: Not addressed; 1: Qualitative discussion; 2: Explicit timing or scheduling analysis | 1 | Determinism increasingly discussed, but quantitative timing analysis remains rare. |
| Reproducibility and tooling transparency | 0: Not reproducible; 1: Partial workflow description; 2: Toolchain and artifacts disclosed | 1–2 | Open tooling improves reproducibility; industrial studies remain opaque. |

- Targeted industrial systems, robotics, or cyber-physical systems involving actuator control or embedded execution.
- Contributed to simulation fidelity, control-oriented modeling, embedded deployment, co-simulation, or real-time execution using FMI-based workflows.
- Provided sufficient technical detail to analyze execution assumptions and interoperability mechanisms.

Publications were excluded if they were purely conceptual, used FMI only peripherally, lacked technical substantiation, or had no implications for embedded or actuator-centric interoperability.

Full-text exclusions were documented using predefined reason codes (see Table 3) to ensure traceability and methodological transparency within the PRISMA-lite screening process.

Reason codes were assigned independently during full-text assessment and resolved by consensus.

To qualify the strength of evidence without artificially filtering an emerging body of work, a lightweight quality assessment was performed on the final set of included studies ($n=36$). Each work was evaluated using a three-level scoring scheme (0–2) across four criteria:

- **Technical specificity:** clarity of FMI interfaces, execution mode, and platform assumptions.
- **Execution evidence:** presence of benchmarks, timing data, HIL validation, or deployment results.
- **Timing and determinism awareness:** whether deadlines, scheduling, jitter/WCET, or clocked execution assumptions are discussed or quantified.
- **Reproducibility signals:** availability of tool versions, configurations, datasets, or implementation details.

This assessment informed interpretation during synthesis, particularly when comparing heterogeneous experimental evidence. Table 4 summarizes the observed trends.

The results highlight a systematic gap between conceptual FMI interoperability claims and fully validated embedded actuator deployments, motivating the need for analyzable execution paths such as eFMI and GALEC.

## 2.4 Data Extraction and Literature Classification

A structured extraction schema (codebook) was defined to ensure consistent classification and to enable auditability of the qualitative synthesis. For each included study, structured data extraction captured the following: FMI version and mode (ME/CS/SE), execution platform (desktop, edge,

MCU, ECU, PLC), real-time assumptions, model characteristics, and reported limitations. Quantitative timing or resource metrics were extracted where available. When timing data were not reported, the absence of quantitative evidence was explicitly recorded to avoid over-interpreting feasibility claims. Extracted fields included:

1. FMI mode and stepping assumptions.
2. Synchronization and scheduling model (if any).
3. Platform class and resource constraints (memory/compute).
4. Actuator-relevant model properties (e.g., stiffness, friction, event density).
5. Reported metrics (e.g., loop rate, latency, jitter, WCET) or validation setting (SIL/HIL/real target).

The full extraction table for the PRISMA-lite corpus is provided as Supplementary S1-A, which has been revised in this version to include, for each of the 36 included studies: (i) FMI version and execution mode (ME/CS/SE); (ii) execution platform class (desktop, edge, MCU/ECU, PLC, HIL); (iii) real-time and scheduling assumptions; (iv) actuator-relevant model characteristics (stiffness, friction, event density); and (v) reported metrics or validation setting (SIL/HIL/real target), with explicit flagging of studies where timing or resource data were absent to avoid over-interpretation of feasibility claims. Bibliographic metadata, source type (peer-reviewed/preprint), and dual-version flags are also included for full traceability. The extended evidence inventory is provided as Supplementary S1-B.

To facilitate the qualitative synthesis, the selected studies were clustered into four primary domains that directly inform the structure of this review:

1. **Standards and Semantics:** Evolution from FMI 2.0 to 3.0.
2. **Actuator Modeling:** High-fidelity requirements and physical dynamics.
3. **Embedded Deployment:** Real-time execution strategies and eFMI code generation.
4. **Tooling Ecosystem:** Industrial readiness and support.

The review emphasizes open standards and documented implementations; proprietary industrial workflows may therefore be underrepresented due to intellectual property constraints. Additionally, as FMI 3.0 and eFMI are recent standards, the literature reflects an ecosystem in transition, with tool support often lagging behind specification features. Variability in reported real-time evidence is addressed by explicitly qualifying conclusions based on the extracted execution assumptions and quality indicators.

## 2.5 PRISMA-Lite Corpus vs. Extended Evidence Inventory

In addition to the PRISMA-lite corpus used for qualitative synthesis, we maintain an Extended Evidence Inventory to support normative and ecosystem-level aspects of the review. PRISMA counts and inclusion decisions apply exclusively to peer-reviewed studies that address actuator-centric FMI usage with embedded/real-time implications. Standards/specifications (e.g., FMI 2.0/3.0, implementers' guides, layered standards), safety regulations (ISO/IEC/RTCA), and tooling/ecosystem resources (tool and library listings, validation/cross-check infrastructure) are included as contextual evidence but are explicitly excluded from PRISMA counts.

This separation avoids inflating the research corpus while enabling triangulation: the PRISMA-lite synthesis (Supplementary S1-A, expanded to $n=36$ after stricter actuator-centric screening of the initial corpus plus integration of 8 new entries from the Web of Science supplementary search) supports RQ1–RQ4 with peer-reviewed evidence, whereas the Extended Evidence Inventory (Supplementary S1-B, which now also includes the 23 entries reclassified from the initial corpus as non-actuator-centric or duplicate, plus 4 new contextual records from the Web of Science supplementary search) documents normative semantics, conformance infrastructure, and tooling maturity that materially affect industrial readiness.

## 2.6 Evidence Snapshot and Review Synthesis

To make the review evidence-weighted beyond narrative discussion, Table 5 summarizes the dominant execution trajectories found in the literature and their typical evidence strength. This snapshot complements the PRISMA-lite selection and the quality assessment, and clarifies how model-level interoperability progressively diverges from execution-level determinism as deployment targets move toward embedded actuator control.

Overall, the evidence indicates a systematic gap between interoperability at the FMU packaging level and interoperability at the execution level, motivating deterministic paths such as eFMI/GALEC for safety-relevant actuator control. Representative evidence for these trajectories is discussed in Sects. 7 and 8.

To avoid treating the 36 PRISMA studies as a homogeneous evidence base, the corpus was re-audited according to the strength of execution evidence. Eight studies provide direct embedded, real-time, HIL, or code-generation evidence; twenty-one primarily support desktop simulation, tooling, domain-specific co-simulation, or prototype integration patterns; and seven are conceptual, survey, formal, or

**Table 5** Evidence snapshot across FMI-based execution trajectories for actuator-centric industrial systems

| Trajectory | Typical target | Evidence pattern | Key implication |
|---|---|---|---|
| Desktop FMI (ME/CS) | Desktop / server | Mature tools; extensive use for MiL/SiL and co-simulation | Strong model-level interoperability, but timing semantics and determinism remain host- and tool-dependent. |
| Runtime FMU on edge/MCU | Edge / MCU | Few prototypes; selective benchmarks; strong platform dependence | Feasible for non-safety-critical local functions (monitoring/estimation), but solver support, footprint, and jitter require careful engineering. |
| FMI 3.0 Scheduled Execution | RTOS / edge | Conceptual evidence only in S1-A for clocks/SE (S1-033, S1-038); no empirical actuator embedded clock/SE benchmark with jitter/WCET/memory tuple | Enables timing-aware integration through clocks/partitions, but industrial readiness is inferred from semantics rather than demonstrated by complete embedded metrics. |
| eFMI/GALEC code generation | ECU / embedded | Industrial pilots; analyzability focus (bounded memory / timing arguments) | Suitable for certification-oriented deployment, at the cost of restricted modeling subsets and higher integration effort. |

specification-oriented studies. No included study reports the complete actuator-relevant metric tuple of jitter, WCET, and memory footprint under realistic embedded load.

| Evidence class | n | S1-A study IDs | Use in synthesis |
|---|---|---|---|
| Direct embedded / RT / HIL / code-generation evidence | 8 | S1-029, S1-034, S1-036, S1-040 [P], S1-044, S1-284, S1-331, S1-351 | Primary evidence for feasibility claims; quantitative reporting remains incomplete. |
| Desktop, tooling, domain simulation, or prototype integration evidence | 21 | S1-028, S1-031, S1-032, S1-037, S1-041, S1-042, S1-045 [P], S1-266, S1-269, S1-277, S1-285, S1-293, S1-311, S1-336, S1-339, S1-350, S1-352, S1-353, S1-354, S1-356, S1-357 | Used to compare interoperability patterns, not to claim hard real-time readiness. |
| Conceptual, survey, formal, or specification-oriented evidence | 7 | S1-030, S1-033, S1-038, S1-039, S1-274, S1-300, S1-355 | Used for semantics, taxonomy, and problem framing. |
| Complete jitter + WCET + memory tuple under embedded actuator load | 0 | None | Absence of complete quantitative reporting is itself a review finding. |

The same audit clarifies FMI 3.0 evidence. Fifteen S1-A entries mention FMI 3.0 or FMI 2.0–3.0 compatibility, but only S1-033 and S1-038 directly address FMI 3.0 clocks or Scheduled Execution, and both are conceptual. Thirteen mixed-version entries report compatibility, tooling, or co-simulation use without isolating empirical FMI 3.0 clock/SE behavior. Therefore, the PRISMA corpus contains zero actuator-centric embedded studies that empirically test FMI 3.0 clocks or Scheduled Execution with reported jitter, WCET, and memory evidence.

| FMI 3.0 evidence category | n | S1-A study IDs | Interpretation |
|---|---|---|---|
| Direct FMI 3.0 semantic studies | 2 | S1-033, S1-038 | Conceptual evidence for clocks and Scheduled Execution, not empirical embedded validation. |
| Mixed FMI 2.0–3.0 compatibility labels | 13 | S1-039, S1-041, S1-045, S1-274, S1-300, S1-311, S1-336, S1-339, S1-352, S1-354, S1-355, S1-356, S1-357 | Compatibility or tooling scope is reported, but FMI 3.0 clock/SE behavior is not isolated. |
| Empirical FMI 3.0 clock/SE tests on actuator embedded workloads | 0 | None | This supports the conclusion that FMI 3.0 execution evidence is still emerging. |
| eFMI/GALEC embedded code-generation evidence | 2 | S1-034, S1-036 | Relevant to analyzable embedded deployment, but distinct from FMI 3.0 SE runtime evidence. |

## 3 Functional Mock-up Interface (FMI) as an Interoperability Standard

FMI is a vendor-neutral open standard that enables the exchange and co-simulation of dynamic system models across heterogeneous tools, platforms, and engineering domains [11]. FMI defines a standardized interface and packaging format, the FMU, that encapsulates model behavior, metadata, and dependencies. Concretely, an FMU packages a machine-readable model description, binaries and/or source code for one or more target platforms, and resources (e.g., tables or parameter files), enabling tool-independent import and execution. By decoupling model definition from its execution environment, FMI supports modularity, traceability, and interoperability across the engineering lifecycle [20, 23, 34], but, critically, interoperability is only as strong as the execution semantics provided by the standard and the master algorithm or scheduler that instantiates them.

From an ecosystem perspective, FMI has become a cornerstone in model-based engineering workflows, with an expanding set of tools for simulation, control, verification, and emerging embedded/edge deployments [34, 35]. Rather than binding interoperability to a specific modeling language, FMI acts as a computational contract between a model component (FMU) and an importing environment (simulation master, co-simulation orchestrator, or real-time scheduler). Representative platforms supporting FMU import/export and/or co-simulation orchestration include Dymola, Simcenter Amesim, MapleSim, SimulationX, Twin Builder, OpenModelica, Simscape, and cloud-native solutions such as Modelon Impact and ODE [36–41]. However, tooling maturity varies substantially across FMI versions and features, particularly for timing-aware semantics and embedded targets.

To make this review explicit about the boundary between standardized interoperability and host-defined execution behavior, Table 6 summarizes what FMI specifies at the interface level versus what typically remains an implementation responsibility (solver, scheduler, and runtime constraints). This distinction is central to the embedded execution gap discussed in Sect. 1.

### 3.1 FMI 2.0: Model Exchange and Co-Simulation in Industrial Applications

FMI 2.0 defines two main usage paradigms, ME and CS [20]. In ME, the FMU provides a system of differential-algebraic equations that are numerically integrated by the importing environment, allowing solver customization and tight coupling with simulation backends. ME therefore shifts numerical responsibility (step-size control, event handling, and error control) to the importer, which is advantageous for stiff actuator models but also implies that reproducibility depends on the selected solver and configuration. In CS, the numerical solver is encapsulated within the FMU, enabling loose coupling between heterogeneous models and tools while protecting vendor intellectual property. In CS, the importing environment typically executes a master algorithm that coordinates communication points and data exchange between FMUs; hence, global determinacy and stability depend on the orchestration policy, coupling strength, and communication step size.

**Table 6** FMI as an interoperability contract: standardized interface elements versus host-defined execution responsibilities

| Aspect | Standardized by FMI | Typically host-defined / tool-dependent |
|---|---|---|
| Model packaging | FMU structure, model description, variables, resources | Binary portability constraints; platform toolchains |
| Numerical execution (ME) | FMI function interface, event indicators, state access | Solver choice, step control, tolerances, event localization policy |
| Co-simulation (CS) | Communication-step abstraction, FMI call sequence | Master algorithm, extrapolation/iteration, rollback support, coupling policy |
| Timing semantics (FMI 3.0) | Clocks, clocked variables, partition concepts | Scheduler mapping, task priorities, deadlines/jitter budgets, timing evidence |
| Embedded deployability | Interface-level support (SE hooks) | Memory policy, WCET characterization, certification arguments |

From an actuator-control perspective, the master algorithm is not an implementation detail; it defines the effective coupling delay, the extrapolation assumptions between communication points, and the admissible handling of discontinuities (e.g., friction transitions, saturation, switching). In strongly coupled or stiff actuator subsystems, these choices can introduce coupling-induced oscillations or phase lag that would not appear under monolithic integration. Consequently, FMI 2.0 CS should be interpreted as providing component interoperability (FMU exchange) while leaving system-level determinacy to orchestrator design. [21, 22]

Co-Simulation FMUs have become the prevalent mechanism for virtual integration in industrial workflows. Simulation environments such as TwinBuilder [42], Simulink with the FMI Blockset [43], and Simcenter Amesim [37] support FMU import/export across toolchains, facilitating system-level validation and supplier integration. Co-simulation is also central in applications such as HIL testing and digital twin pipelines. Nevertheless, from a computational perspective, FMI 2.0 CS provides limited standardized guidance on time coordination beyond the communication-step abstraction; this is a known source of numerical drift, instability for strongly coupled subsystems, and run-to-run variability when solver and step policies change [21, 22].

In the actuator control domain, FMI 2.0 has been employed for the validation of embedded control units and integration of high-fidelity subsystem models within co-simulation workflows [17]. Furthermore, eFMI has been proposed to support deterministic code generation for embedded targets [25]. In industrial workflows, FMUs are commonly used for supplier model exchange and system-level integration under IP constraints; reported applications include SIL/HIL validation setups and distributed co-simulation across organizational boundaries.

However, FMI 2.0 has notable limitations for embedded deployment and hard real-time integration. The standard does not provide native semantics for deterministic task scheduling, event timing, or priority-aware execution, which are essential for control under RTOS constraints and microcontroller-class targets [12, 16]. In practice, FMI 2.0 enables model-level interoperability, but it under-specifies execution-level interoperability: memory allocation behavior, timing contracts, WCET considerations, and scheduling

interfaces are outside the standard. As a consequence, execution-level claims (e.g., real-time feasible) must be interpreted cautiously unless accompanied by platform disclosure and quantitative timing evidence. As a result, deploying FMUs on constrained hardware typically requires custom wrappers, static linking strategies, or model simplification, which can reduce fidelity or increase engineering effort. Despite these gaps, FMI 2.0 remains the most widely supported version and forms the baseline for interoperability in simulation and control across industrial sectors [35, 39, 44].

### 3.2 Evolution Toward FMI 3.0 and Embedded-Oriented Capabilities

FMI 3.0 extends FMI 2.0 to better support embedded, event-driven, and real-time domains [23]. The key shift is a stronger semantic vocabulary for time and execution, enabling importing environments to represent sampled-data behavior and to coordinate computation in a scheduler-driven manner. In particular, the FMI 3.0 line introduces semantic constructs that directly address timing and scheduling limitations of prior versions [33, 45, 46]. The most notable enhancements include:

- **Clocks and Clocked Variables**: explicit representation of discrete-time, sampled, and event-driven behavior within FMUs, supporting control loops that operate at fixed sampling rates or depend on asynchronous triggers [23, 24, 33].
- **Scheduled Execution**: an execution interface that enables explicit partitioning of computations and delegation of when model partitions execute to an external scheduler. This aligns FMI integration with RTOS-based and deterministic scheduling policies used in embedded control [23, 24, 33], and provides a pathway to map model computations to periodic/aperiodic tasks with explicit activation semantics.
- **Extended Data Types and Streaming**: interface extensions that improve data exchange efficiency and broaden applicability to heterogeneous and tool-integrated workflows [23, 45], which is particularly relevant for cyber–physical pipelines where bandwidth and serialization overhead can dominate end-to-end latency.

These features address the needs of real-time actuator systems, where sampling, latency, and execution determinism are tightly constrained. For FMI 3.0 specifically, clocks and Scheduled Execution provide the semantic foundation for making timing assumptions explicit at the interface level [23, 24]. For deployment-oriented code generation, eFMI/GALEC evidence is discussed separately because it concerns analyzable generated artifacts rather than FMI 3.0 SE runtime behavior [25, 26].

A practical interpretation of FMI 3.0 Scheduled Execution is that it exposes model computations as scheduler-activated partitions. This makes it conceptually compatible with RTOS execution models (periodic control tasks, sporadic event handlers, and partition dependencies). Nevertheless, SE does not by itself provide WCET bounds, memory policies, or schedulability arguments; these remain properties of the generated/implemented FMU code and the importing runtime. Therefore, SE should be viewed as a semantic enabler of timing-aware interoperability, rather than a complete, certification-ready execution framework.

It is important to note that the tooling ecosystem is beginning to incorporate FMI 3.0 concepts, but support remains mixed across features and products. Public ecosystem statistics and vendor/tool listings indicate growing uptake of FMI 3.0 alongside the still-dominant FMI 2.0 base [35]. At the same time, emerging integration tools (e.g., FMI-to-automation bridges and ROS 2 integration layers) illustrate early paths toward cyber–physical deployment and controller-in-the-loop experimentation [47, 48], although feature completeness for scheduled execution and clock semantics is still uneven across toolchains.

However, industrial adoption of FMI 3.0 for embedded workflows remains limited compared to FMI 2.0 due to concerns about toolchain stability, incomplete feature coverage, and integration costs. Consequently, FMI 3.0 usage is currently most common in research prototypes, pilot deployments, and tightly controlled toolchains [33, 49].

From a system-level perspective, FMI provides a robust and evolving foundation for tool interoperability in high-fidelity actuator simulation. FMI 2.0 enables modular model exchange and co-simulation in industrial workflows, while FMI 3.0 introduces timing-aware semantics and scheduling mechanisms required for embedded and real-time execution. This evolution directly targets the embedded execution gap by moving from tool connectivity toward explicit execution semantics; nevertheless, achieving deployable embedded interoperability typically requires additional constraints (e.g., bounded memory, WCET characterization, and certification-oriented code generation), motivating embedded-focused approaches such as eFMI and GALEC.

## 4 High-Fidelity Simulation Requirements for Actuator Control

High-fidelity simulation is a key enabler in the development, validation, and deployment of industrial actuator control systems. In contrast to the simplified linear models often used in early-stage controller design, industrial-grade

control requires models that capture complex nonlinear dynamics, timing effects, and hardware interactions, particularly when actuators operate near physical limits or under tight bandwidth and safety constraints [1, 4, 5]. For actuator-centric loops, high-fidelity is not synonymous with high order; it denotes the ability to reproduce the dominant nonlinearities and timing-sensitive effects that shape closed-loop stability, safety margins, and performance in the target operating regime. This section defines the requirements for high-fidelity modeling in actuator-centric applications and analyzes how FMI-based encapsulation supports model reuse across simulation, testing, and (increasingly) embedded integration. To make the requirements operational (and to connect them to execution-level interoperability), we explicitly map actuator phenomena to their dominant numerical consequences and to the FMI execution implications (ME/CS/SE and embedded deployability).

### 4.1 High-Fidelity in Industrial Actuators

In industrial contexts, fidelity is not defined only by model complexity, but by a model's ability to reproduce the dominant phenomena that influence stability margins, safety envelopes, and performance under realistic operating conditions. For actuators, this typically requires capturing multi-domain effects (mechanical, electrical, thermal) and discontinuities that strongly affect closed-loop behavior [3, 50]. From a computational perspective, actuator fidelity is often coupled to numerical properties: friction and contact introduce discontinuities and event density; elastic elements can induce stiffness; and power/drive dynamics add multi-rate behavior. These properties directly affect solver choice, step-size constraints, and real-time feasibility. Key aspects include:

- **Nonlinear dynamics**: saturation, friction, backlash, dead zones, and hysteresis; friction alone often induces limit cycles and tracking degradation if not modeled and compensated [6, 7]. In simulation, friction and backlash can yield discontinuous right-hand sides and frequent mode switches, increasing event handling overhead and sensitivity to step-size and solver tolerances.
- **Compliance and resonance**: including intentional elasticity in Series Elastic Actuators (SEAs) and Variable Stiffness/Impedance Actuators (VSAs/VIAs), which directly shape achievable bandwidth and interaction safety [2, 8, 51]. Compliance can create fast/slow time-scale separation; high stiffness and resonant modes can render models numerically stiff, requiring implicit integration or small steps to preserve stability.
- **Power, drive, and thermal effects**: current limits, thermal coupling, switching artifacts, and PWM-related losses in electric drives, which can introduce nonlinear constraints and slow dynamics relevant to embedded control and protection functions [3, 4]. Drive models are frequently multi-rate (fast electrical dynamics, slower thermal dynamics) and may require partitioning or reduced-order representations for real-time execution.
- **Timing and computation effects**: control delays, sampling behavior, quantization and discretization artifacts, and task scheduling jitter, all of which impact embedded controller performance and may dominate in high-rate servo loops [5, 9, 10]. These effects are not merely implementation details: they define execution semantics that can change the closed-loop response, and must be modeled explicitly when validating embedded controllers or certifying timing behavior.

Table 7 summarizes this mapping from actuator phenomena to numerical consequences and execution implications. The table is intended as a compact lens for requirements-to-execution throughout the review, covering FMI modes, scheduled execution, and embedded deployability.

These characteristics are increasingly relevant in actuator subsystems operating near mechanical or thermal constraints, especially in safety-critical robotics, collaborative automation, and high-bandwidth servo control. In such regimes, fidelity must be assessed relative to the intended use case (e.g., controller tuning, HIL/VC validation, fault detection), not only by model order. Accordingly, this review treats actuator fidelity as a requirement vector:

**Table 7** Actuator fidelity drivers mapped to numerical consequences and interoperability implications for FMI-based workflows

| Actuator phenomenon | Dominant numerical consequence | Interoperability / execution implication |
|---|---|---|
| Friction, backlash, saturation | Discontinuities; high event density; solver sensitivity | CS master must support event-rich coupling; ME reproducibility depends on solver/tolerances; embedded feasibility depends on bounded event-handling cost. |
| Compliance and resonant modes | Stiffness; time-scale separation; small-step requirements | ME often preferred for stiff integration; CS requires conservative communication steps; SE requires explicit activation and bounded partition time. |
| Drive electronics and multi-rate dynamics | Multi-rate behavior; mixed fast/slow states; potential algebraic loops | Partitioning or reduced-order models may be required; SE/clocks represent sampled-data partitions; embedded targets may impose code-generation constraints. |
| Sampling, delay, jitter | Phase lag and timing variability; performance degradation | Timing must be explicit: SE/clocks, RTOS mapping, and evidence (latency/jitter/WCET) become first-class artifacts. |

1. Physical nonlinearities.
2. Numerical stiffness/event density.
3. Timing semantics (sampling, delay, and jitter).

This triplet provides a direct bridge to interoperability discussions in later sections (FMI execution modes, scheduled execution, and embedded code generation). As a practical consequence, fidelity should be reported together with the numerical/execution assumptions under which it was validated (solver/step policy, event tolerances, and communication-step settings); otherwise, cross-tool comparability is weak.

### 4.2 Model-Based Actuator Simulation Using FMI

Model-based approaches provide a structured methodology for designing, analyzing, and validating actuator systems. FMI extends this methodology by enabling encapsulation of high-fidelity, tool-independent actuator models for reuse across simulation, testing, and system integration stages [11, 20]. In actuator-centric contexts, the practical value of FMI is interface stability: validated plant components can be integrated into different orchestrators (desktop co-simulation, virtual commissioning, HIL benches, and controller-in-the-loop pipelines) without re-implementing the plant model.

By exporting a validated actuator model as FMU, engineers can integrate it into virtual commissioning workflows, system-level co-simulation, and HIL testbenches while keeping model interfaces stable across tools and organizations [25, 26]. In robotics and cyber-physical experimentation, FMI-compatible integration layers such as `gazebo-fmi` [52], and ROS 2 FMI adapters [48] enable injecting realistic actuator dynamics into control loops to test controller interactions under more faithful plant behavior [28, 30, 53]. These approaches are particularly valuable when actuator dynamics (friction, compliance, drive saturation) must be exercised under realistic sensor/actuator sampling and control loop timing.

FMI also supports multi-domain composition: models can combine mechanical elements (e.g., joint compliance), electrical subsystems (e.g., motor and drive circuits), and software-like control blocks [54, 55]. However, composition increases computational coupling and may amplify stiffness or event interactions across subsystems. In co-simulation, this often forces conservative communication-step choices; in embedded settings, it often forces model restructuring or partitioning.

Commercial and open-source tools such as SimulationX, Dymola, and MapleSim provide actuator-oriented libraries and support FMU export at different levels of abstraction [36, 38, 39]. However, the ability to export an FMU does not, by itself, guarantee that the exported component can run deterministically or within real-time budgets: numerical stiffness, event handling behavior, and memory allocation patterns can make high-fidelity FMUs impractical for embedded execution without additional constraints.

To preserve traceability while respecting execution constraints, actuator FMUs are often used in a tiered form: a full-fidelity model for design and offline validation; a real-time-capable variant for HIL/closed-loop experimentation under bounded-step policies; and a deployable artifact for embedded targets (often reduced or code-generated under restricted semantics). The central interoperability requirement is that these tiers remain linked by explicit validation tests and documented numerical/execution assumptions.

From a control engineering standpoint, high-fidelity FMUs can coexist with reduced-order or deployment-oriented models within a unified workflow. For example, full nonlinear drive models may be used during design and verification, while reduced or code-generated variants (e.g., via eFMI/GALEC restrictions) can be deployed on embedded platforms [12, 25, 27]. Domain-specific co-simulation studies confirm this pattern across diverse actuator domains: ship electric propulsion drives [56], wind turbine drive-trains [57], and multibody structural assemblies with compliant actuators [55] all demonstrate multi-physics FMI integration where fidelity requirements drive the coupling strategy. This motivates a fidelity–feasibility trade-off: high fidelity improves predictive validity, whereas deployment requires bounded compute/memory and stable timing. Interoperability standards are therefore most impactful when they preserve traceability between the validated high-fidelity model and the constrained executable artifact used on the embedded target.

However, fidelity comes at the cost of computational complexity. Large FMUs with stiff or high-order dynamics may require real-time solvers, careful step-size control, or model restructuring for embedded use. As shown in embedded/edge benchmarking studies, platform capabilities vary widely, and practical deployments often require balancing fidelity against real-time feasibility [16]. When the actuator model is event-rich or stiff, naive co-simulation stepping can lead to numerical instability or non-deterministic execution times, which is unacceptable for hard real-time control loops. This motivates complementary techniques such as model reduction, numerical tuning, and explicit timing semantics (e.g., clocks and Scheduled Execution in FMI 3.0) to better align simulation models with real-time integration requirements [23, 24, 33], as well as analyzable code-generation workflows (eFMI/GALEC) for certification-oriented contexts.

From an engineering standpoint, FMI-based modeling provides a practical mechanism to unify actuator simulation

across engineering phases and deployment targets. Its main value in actuator-centric workflows lies in preserving stable interfaces and enabling cross-tool reuse of validated models, while emerging embedded-oriented approaches (FMI 3.0 timing semantics and eFMI code generation) address the determinism and resource constraints that limit direct deployment of high-fidelity models on constrained hardware. Table 8 distinguishes evidence basis, reported execution metrics, and synthesis role, so that tool availability claims are not conflated with quantitative evidence of real-time feasibility. NR denotes not reported in the reviewed source.

## 5 Interoperability Across the Industrial Control Stack

Interoperability challenges in industrial systems extend beyond model authoring and affect the entire control stack from high-level simulation and virtual commissioning environments to embedded controllers, industrial networks/field buses, and actuator hardware. Actuator control systems are particularly exposed because they must integrate physical plant models, control algorithms, communication protocols, and execution platforms under stringent timing, safety,

**Table 8** Representative actuator and drive-focused studies supporting FMI-based high-fidelity simulation and actuator control workflows

| Reference | Evidence basis | FMI scope | Reported execution metrics | Evidence strength | Use in synthesis |
|---|---|---|---|---|---|
| Bosch (via ROS2 FMI Adapter) [48] | Tool documentation / integration layer | FMI 2.0 | NR for jitter/WCET/memory | Tool availability | Shows feasible ROS 2 connectivity for actuator FMUs, not quantitative real-time readiness. |
| Eklund et al. (2021) [58] | Peer-reviewed simulation study | FMI (workflow-dependent) | NR for jitter/WCET/memory | Domain simulation | Supports drive-level model reuse and lifecycle analysis, not embedded deployment. |
| Arias et al. (2022) [28] | Robotics co-simulation prototype | FMI 2.0 (CS) | Simulation-loop timing discussed; no WCET or memory tuple | Prototype | Supports actuator-dynamics injection in robotics validation loops. |
| robotology / gazebo-fmi [52] | Open-source tooling | FMI 2.0 | NR for jitter/WCET/memory | Tool availability | Useful as ecosystem support; not treated as empirical performance evidence. |
| Lange et al. (2025) [53] | Middleware integration prototype | FMI (integration) | Loop-timing issues discussed; no complete quantitative tuple | Prototype | Supports the middleware-latency argument for controller-in-the-loop workflows. |
| Oldemeyer et al. (2024) [30] | Robot-controller system prototype | FMI 2.0 (CS) | RT assumptions described qualitatively; no jitter/WCET/memory tuple | System prototype | Supports modular actuator-controller composition in ROS 2. |
| Picerno et al. (2022) [59] | HIL validation | FMI 2.0 (CS) | Real-time HIL setting reported; complete tuple NR | Direct RT/HIL evidence | Supports real-time validation use, but not general FMU deployment guarantees. |
| Perabo et al. (2025) [56] | Multiphysics drive simulation / prototype | FMI 2.0–3.0 (CS) | NR for embedded jitter/WCET/memory | Domain simulation | Supports multi-domain drive coupling, not FMI 3.0 SE evidence. |
| da Silva et al. (2023) [57] | Drive-train co-simulation | FMI 2.0 (CS) | NR for jitter/WCET/memory | Domain simulation | Supports actuator-relevant electromechanical coupling patterns. |
| Serban et al. (2026) [55] | Multibody solver benchmark / integration | FMI 2.0–3.0 (ME/CS) | Benchmark-oriented; not embedded RT-focused | Tool/ simulation benchmark | Supports FMI integration for compliance-rich multibody dynamics. |
| Klemets et al. (2026) [31] | Open-source in-the-loop software | FMI 2.0–3.0 (CS) | NR for embedded jitter/WCET/memory | Tooling evidence | Supports reproducible in-the-loop experimentation, not hard real-time claims. |
| Ultsch et al. (2025) [60] | Simulation / software framework | FMI 2.0–3.0 (CS) | NR for real-time execution metrics | Software evidence | Supports FMU-based learning workflows; excluded from embedded feasibility claims. |

and reliability constraints [16, 18, 19, 26]. In this review, interoperability is treated as a multi-layer property:

1. Syntactic compatibility (interfaces and data types).
2. Semantic compatibility (model meaning and stepping assumptions).
3. temporal/execution compatibility (scheduling, latency, jitter, and resource bounds).

This section examines how the FMI standard supports interoperability across such heterogeneous environments and discusses its role in bridging traditionally siloed engineering workflows. To make this multi-layer view operational, Table 9 maps representative stack layers to their typical interoperability artifacts and to the execution-level bottlenecks that often dominate actuator-centric deployments.

Traditional industrial control architectures rely on vertically integrated toolchains centered around PLC programming environments (IEC 61131–3), vendor-specific drive configurators, and proprietary modeling/simulation tools. While optimized for individual ecosystems, these workflows tend to restrict cross-platform model reuse and complicate cross-vendor integration. As a result, actuator models developed in simulation tools are often not reused in real-time controllers, leading to reimplementation, model drift, and increased validation cost [13–15]. This break in continuity is most visible at the simulation-to-deployment transition: models validated with desktop solvers and flexible timing assumptions are re-implemented under RTOS/PLC execution semantics, where determinism and bounded resources dominate. In actuator-centric systems, this discontinuity is amplified because tight sampling periods and high stiffness/event density make both numerical and temporal semantics central to closed-loop stability.

FMI addresses this issue by offering a standardized interface that decouples model definition from execution environments [11]. By packaging component behavior as FMUs, actuator dynamics and (in some workflows) control-related functionality can be reused across simulation stages, HIL validation, and increasingly deployment-oriented embedded integration [17, 20, 25]. At the model level, FMI provides a portable component boundary; at the system level, however, end-to-end behavior depends on orchestration policies (master algorithms), communication step sizes, and scheduler integration. Industrial toolchains such as the FMI Blockset for Simulink [43], and FMI-enabled workflows in Twin Builder [42] illustrate how models can be exchanged across system layers while preserving interface compatibility and enabling supplier integration under IP constraints.

At the automation level, emerging integration paths connect FMI models to industrial runtime platforms and programming standards. For example, FMI Terminal Block provides a bridge between FMI models and IEC 61499 function blocks [47], while translation workflows such as CDL–to–PLC demonstrate how model-based designs can be mapped into PLC-compatible implementations [49]. These pathways illustrate a recurring engineering pattern: FMI preserves model modularity and interface stability, while separate compilation or translation steps attempt to recover execution-level guarantees required by PLC/RTOS runtimes. In other words, interoperability is achieved in two stages:

1. A portable model boundary.
2. A deployment-specific recovery of timing/memory guarantees.

These approaches reduce the semantic gap between offline models and operational control logic by maintaining consistent interfaces and facilitating migration from virtual commissioning to industrial controllers.

FMI can also serve as a bridging abstraction in hybrid architectures combining simulation models, ROS controllers, and industrial automation protocols. FMI adapters for ROS 2 [48] and Gazebo integration layers [52] enable actuator-level FMUs to participate in closed-loop experiments and controller-in-the-loop testing, supporting both design-phase

**Table 9** Operational mapping of interoperability across the industrial control stack: artifacts, bottlenecks, and typical mitigations for actuator-centric workflows

| Stack layer | Interoperability artifact | Execution bottleneck | Typical mitigation |
|---|---|---|---|
| Model authoring / plant design | Model + FMU export (ME/CS) | Solver dependence; event density; stiffness | Solver disclosure; validation tests; reduced/partitioned variants. |
| System integration / co-simulation | Orchestrator + master algorithm | Communication-step error; determinacy; coupling delay | Conservative step policies; determinacy rules; rollback/iteration where available. |
| Virtual commissioning / HIL–SIL | FMU-in-the-loop benches | Real-time step budgets; I/O latency; platform overhead | Fixed-step real-time execution; profiling; co-location of latency-critical components. |
| Embedded control (RTOS/ECU/PLC) | Scheduler tasks / code-generation artifacts | WCET/jitter; bounded memory; certification evidence | FMI 3.0 SE hooks; eFMI/GALEC code generation; static allocation policies. |
| Industrial networks / middleware | Fieldbus / middleware timing | End-to-end latency; clock synchronization; serialization overhead | Time-aware scheduling; timestamping; explicit latency budgets. |

integration and runtime experimentation in cyber–physical settings [28, 53]. From an interoperability standpoint, these hybrid pipelines are valuable because they stress timing semantics: actuator dynamics (often stiff and event-rich) are executed alongside controllers with fixed-rate sampling and non-negligible middleware latency, revealing whether co-simulation orchestration preserves stability and determinacy [61]. This is also where execution-level interoperability often fails first: a pipeline may run yet remain unsuitable for actuator control due to unbounded latency spikes, unstable coupling, or host-dependent trajectories.

Digital twins aim to enable continuous interaction between a physical system and its virtual representation to support performance monitoring, predictive maintenance, and closed-loop optimization across the lifecycle [14, 15]. In actuator-centric applications, these twins require high-fidelity models that reflect multi-domain interactions (mechanical–electrical–thermal) and can be synchronized with live control systems, sensors, and operational constraints [16, 28]. Consequently, digital twin interoperability is not only a question of model exchange, but of consistent execution semantics across phases (design, commissioning, operation) and across compute tiers (desktop, edge, embedded).

FMI supports lifecycle-oriented digital twin architectures by enabling validated models to be reused across design, commissioning, deployment, and operation phases [11, 25]. Typical lifecycle patterns include:

- **Design and verification**: actuator FMUs used for control design and performance assessment in multi-domain tools (e.g., Amesim, MapleSim, Dymola) [36–38];
- **Virtual commissioning and integration testing**: co-simulation of actuator FMUs with control logic via FMI Blockset, orchestration frameworks, or domain-specific integration layers [42, 43];
- **Deployment-oriented validation**: reuse of FMU-based reference models in HIL/SIL setups or embedded validation pipelines, including ECU-oriented workflows [17, 59];
- **Embedded/edge deployment paths**: reduced or compiled variants for embedded controllers (e.g., eFMI-to-AUTOSAR) to satisfy determinism and certification constraints [25, 26].

This continuity supports traceability between simulation assumptions and operational behavior, while enabling cross-vendor subsystem composition and IP protection through black-box FMUs [12, 20]. However, FMI alone does not resolve all integration challenges. In networked control stacks, end-to-end timing is shaped by communication latency, clock synchronization, middleware scheduling, and the interaction between controller sampling and FMU stepping. These factors can dominate closed-loop behavior even when model interfaces are fully compatible. Therefore, a robust industrial workflow typically pairs FMI-based model exchange with explicit timing policies (communication-step bounds, sampling/deadline contracts) and with profiling/validation that quantifies latency and jitter under realistic load. FMI 3.0 introduces clocks and Scheduled Execution as semantic building blocks to better align model execution with deterministic timing requirements, but end-to-end feasibility depends on toolchain maturity and integration with real-time schedulers [23, 24, 33].

Overall, FMI provides a unifying interface that enables cross-domain interoperability across the industrial control stack. In actuator-centric applications, this interoperability is particularly valuable when model reuse, timing consistency, and deployment-oriented validation are required. However, FMI primarily guarantees model-level interoperability; achieving execution-level interoperability (deterministic timing, bounded memory, and certifiable analyzability) typically requires additional constraints and complementary runtime mechanisms. This is precisely why the review separates syntactic/semantic integration success from evidence of real-time feasibility.

Figure 3 operationalizes this distinction by systematically mapping the transition from desktop validation workflows to embedded deployment strategies along a stringency axis, and positioning each trajectory within the toolchain readiness framework (R0–R3) introduced in Sect. 8.3. It illustrates how FMUs serve as a common interoperability artifact across simulation environments, while execution semantics and temporal guarantees progressively diverge as deployment targets shift toward resource-constrained, certification-driven embedded platforms. The figure thereby serves as a navigation aid for the remainder of the review: each trajectory is discussed in terms of its execution contract, evidence strength, and fit for actuator-centric use cases.

The left domain represents simulation and validation workflows where FMI-based FMU exchange is mature and widely adopted. The right domain highlights embedded deployment strategies, distinguishing between runtime-constrained FMU reuse and eFMI/GALEC-based deterministic code generation. The figure emphasizes how execution semantics and analyzability requirements increasingly dominate as systems transition toward production-grade embedded control loops, thereby motivating the execution-focused review in Sects. 6, 7, and 8.

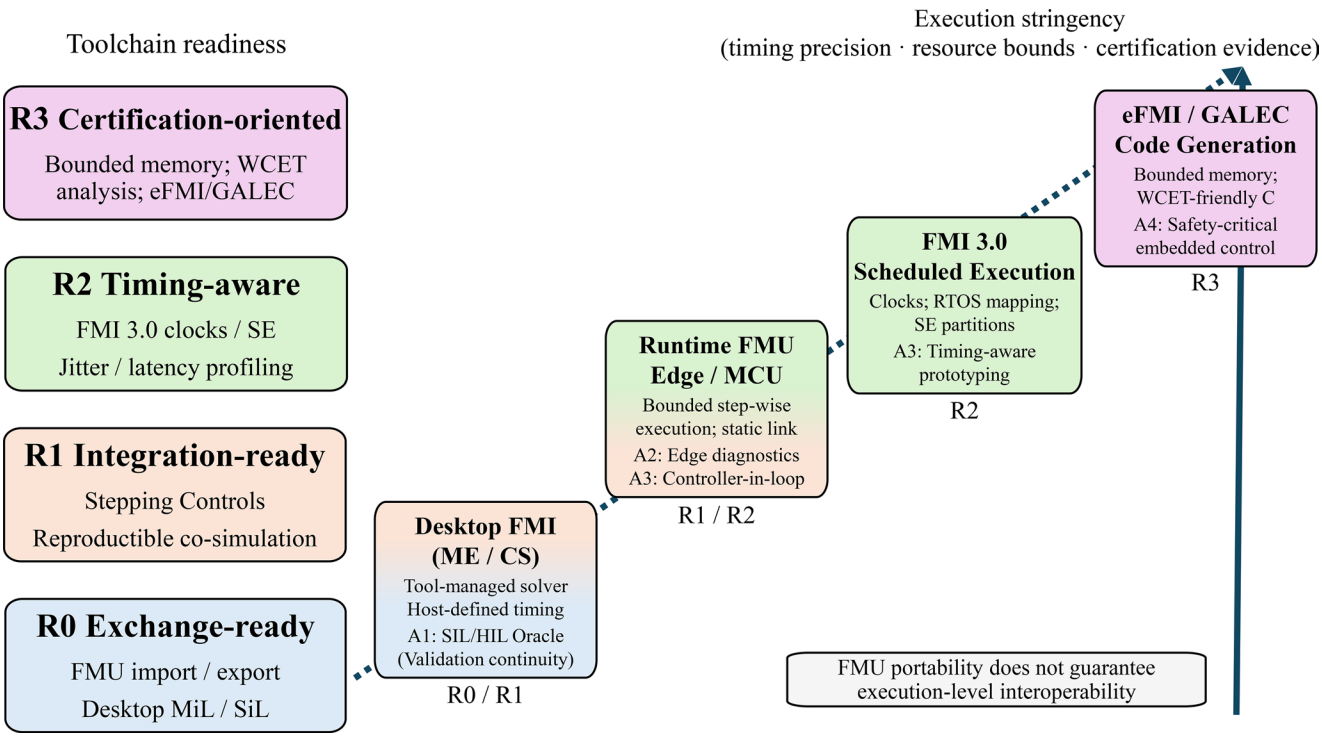



**Fig. 3** Execution and deployment trajectories of FMI-based models in actuator-centric industrial systems, mapped across increasing execution-level stringency (left to right). Four trajectories are shown and positioned according to the toolchain readiness levels (R0–R3) defined in Sect. 8.3: desktop ME/CS (R0/R1, tool-managed integration, low determinism guarantees); runtime FMU on edge/MCU (R1/R2, bounded stepping, medium determinism); FMI 3.0 Scheduled execution on RTOS (R2, timing-aware activation, medium-high determinism); and eFMI/GALEC code generation (R3, bounded resources, certification-oriented). Shaded regions indicate where each trajectory is best suited as a function of timing stringency and resource constraints. The figure makes explicit that model-level FMU portability (left domain) does not imply execution-level interoperability (right domain), and that bridging the embedded execution gap requires progressively stronger execution contracts as deployment moves toward safety-critical embedded control

## 6 Computational Determinism and Real-Time Execution Contracts for FMI

While FMI has matured as a model-exchange and co-simulation interface, industrial deployment hinges on computational determinism: the ability to reason about numerical behavior, timing, and resource usage across heterogeneous execution back-ends. This section consolidates the computational mechanisms that dominate execution-level interoperability and formalizes the execution contracts required to make FMI-based models credible for embedded actuator control. In this review, determinism is treated as a three-part property: numerical determinism (repeatable trajectories under a stated stepping/solver policy), temporal determinism (bounded activation and completion under scheduling constraints), and resource determinism (bounded memory and bounded worst-case execution paths). The embedded execution gap can be interpreted as a failure to preserve one or more of these determinism dimensions during the transition from desktop validation to embedded execution.

### 6.1 Co-Simulation Master Algorithms, Numerical Stability, and Determinate Composition

In FMI co-simulation (CS), the importing environment orchestrates multiple FMUs using a master algorithm that schedules communication points, propagates inputs/outputs, and resolves discontinuities. The choice of master algorithm is not a cosmetic detail; it defines the numerical contract between FMUs (step size, extrapolation strategy, rollback capability, and event handling) and, therefore, determines whether actuator-relevant closed-loop behavior remains stable and reproducible. A central challenge is that high-bandwidth actuator loops amplify synchronization errors. When stiff dynamics, friction discontinuities, saturation, or switching power electronics are present, the coupling error at communication points can dominate performance and even destabilize the loop if the orchestration semantics are not aligned with the actuator sampling regime. These effects are well known in the co-simulation literature and motivate determinate composition principles, conservative step control, and explicit handling of event-dense trajectories [21, 22, 62, 63].

From a computational standpoint, three master-algorithm features become decisive for actuator-centric use cases:

- **Communication-step control:** fixed-step vs. adaptive step strategies, and whether step sizes can be bounded by the fastest actuator sampling period.
- **Event and discontinuity management:** detection and localization of events across FMUs, including the handling of simultaneous events that can break determinacy.
- **Rollback/iteration support:** whether the master and FMUs can negotiate rollback to improve coupling consistency (often required for stiff or strongly coupled subsystems).

In practice, many industrial co-simulation workflows adopt conservative communication-step policies (often fixed-step at the controller sampling rate) when actuator loops are involved, because stability margins are sensitive to coupling delays and extrapolation errors. This design choice is computationally conservative but improves predictability and debuggability, which is critical for validation and regression testing. A key implication is that successful co-simulation is not equivalent to execution-level interoperability: the latter requires that the chosen master policy be disclosed and that the resulting trajectories be reproducible under controlled step policies.

To make this review evidence-weighted, Table 10 summarizes typical master-algorithm failure modes and mitigation strategies that recur across FMI co-simulation studies and tool reports.

To avoid over-claiming determinacy, co-simulation studies should report a minimal reproducibility envelope: the communication-step policy (fixed/adaptive, maximum step), whether extrapolation is used between communication points, whether rollback/iteration is enabled, and the event-handling strategy used by the master. Without these disclosures, observed actuator performance may be an artifact of an implicit orchestrator policy rather than a transferable property of the FMU model.

### 6.2 From FMI 3.0 Scheduled Execution to RTOS Task Models

FMI 3.0 introduces clocks and Scheduled Execution (SE) to expose timing-aware activation points and partitioned computation to an external scheduler [23, 24, 33]. Conceptually, SE provides semantic hooks for mapping FMI execution into RTOS task models, but those hooks are not themselves timing guarantees. Moving from SE semantics to embedded credibility requires explicit execution contracts: period/deadline constraints, WCET envelopes, memory policies, and jitter budgets that can be validated under realistic interference [9, 10].

For actuator-centric embedded control, a minimal execution contract can be expressed as:

- **Activation contract:** sampling period $T_s$, deadline $D \leq T_s$, and allowable jitter $J$ for each clocked partition.
- **Computation contract:** WCET estimate or bound per partition (including worst-case branches triggered by events or saturation).
- **Resource contract:** bounded memory (no unbounded allocation), bounded stack usage, and bounded I/O serialization overhead.
- **I/O contract:** data freshness assumptions and coupling latency bounds across bus/middleware boundaries.

These contracts are standard practice in embedded real-time engineering but are rarely reported systematically in FMI-based studies. Consequently, interoperability at the FMU packaging level does not automatically translate into interoperability at the execution level. This review, therefore, recommends that embedded-oriented FMI studies report activation parameters, timing measurements under load, and resource budgets as first-class artifacts.

Table 11 provides a compact reporting checklist that can be used by authors and tool vendors to strengthen execution-level claims.

Execution contracts are most useful when they are paired with observable failure signatures. For actuator-centric

**Table 10** Computational failure modes in FMI co-simulation and practical mitigation strategies for actuator-centric loops

| Failure mode | Typical cause | Mitigation (computational contract) |
|---|---|---|
| Coupling-induced oscillations | Communication step too large; extrapolation error; strong stiffness | Bound communication step by actuator sampling rate; prefer fixed-step at loop rate; reduce extrapolation order; enable iteration/rollback when available. |
| Non-reproducible trajectories | Host-dependent solver settings; hidden tolerances; inconsistent event ordering | Report step-size bounds and event tolerances; use determinate composition rules; log orchestrator configuration for regression testing. |
| Event thrashing / Zeno-like behavior | High event density (friction/switching); poor event localization | Introduce hysteresis or regularization where physically justified; constrain event-localization iterations; synchronize event clocks across components. |
| Latency inflation in closed-loop | Orchestrator overhead; excessive iterations; I/O serialization | Profile orchestrator and FMU calls; minimize coupling iterations; co-locate latency-critical components; use streaming data types where supported. |

**Table 11** Reporting checklist for execution-level interoperability of FMI-based actuator workloads (recommended for embedded/SE studies)

| Contract item | What to report | Why it matters |
|---|---|---|
| Activation | $T_s$, $D$, jitter budget; clock type | Defines control-loop feasibility and sampling stability. |
| Timing evidence | WCET distribution, jitter envelope, load conditions | Separates feasibility claims from demonstrable execution evidence. |
| Solver semantics | Step policy, tolerances, event-handling strategy | Prevents host-dependent trajectories and irreproducible behavior. |
| Memory policy | Static vs. dynamic allocation; peak RAM/stack | Critical for safety/certification alignment; avoids runtime failures. |
| I/O latency | Bus/middleware latency; serialization overhead | Execution-level interoperability often breaks at I/O boundaries. |
| Toolchain disclosure | Versions, configuration, scripts | Enables reproducibility and regression testing. |

workloads, common contract violations include: deadline misses (task overruns beyond $D$), jitter bursts exceeding $J$ that destabilize the loop, unbounded event-handling paths that cause timing outliers, and unexpected dynamic memory allocations that break resource determinism. These signatures are measurable in practice (e.g., timestamped activation/completion logs, high-water memory marks) and should be reported alongside qualitative feasibility claims.

Finally, the relationship between SE semantics and classic schedulability theory is under-exploited in current FMI toolchains. Priority assignment methods and schedulability tests (e.g., fixed-priority analysis) are well established, but their systematic integration with FMI-based partitions remains largely ad hoc. This gap motivates embedded-oriented FMI profiles that bundle SE semantics with scheduling metadata and validation suites [9, 10, 64].

## 7 Embedded Systems and Edge Execution of FMI Models

Deploying high-fidelity models on embedded and edge platforms represents one of the most challenging and impactful frontiers for FMI-based interoperability in industrial applications. While FMI has traditionally been applied in desktop and server-class simulation environments, industrial automation trends increasingly push computation closer to actuator hardware to enable low-latency control, local autonomy, and resilient operation under resource and real-time constraints [16]. Consequently, the feasibility of executing FMI-based models on embedded targets becomes a key enabler for next-generation actuator systems and deployment-oriented validation workflows. From a computational viewpoint, embedded deployment requires that an FMU (or its derived artifact) satisfies an execution contract: bounded memory, bounded execution time per step, explicit stepping/scheduling semantics, and predictable numerical behavior under fixed sampling. This section therefore focuses on execution feasibility evidence rather than on FMU portability alone.

### 7.1 Challenges of Running High-Fidelity Models on Embedded Hardware

Embedded controllers used in industrial actuators are typically constrained in memory, processing power, and energy availability. Common targets range from microcontroller-class devices (e.g., Cortex-M) to SoC-based controllers running real-time operating systems (RTOS). In these environments, engineering constraints such as static memory allocation policies, limited floating-point throughput, and hard sampling deadlines shape what can be executed reliably in closed loop [9, 16]. High-fidelity actuator models incorporating nonlinearities, compliance, and multi-domain dynamics quickly challenge such platforms, especially under tight control-loop deadlines. In addition to average compute load, worst-case behavior (event bursts, mode switches, cache effects) can dominate real-time feasibility and must be considered when actuator loops operate at kHz rates.

A primary bottleneck is computational load. Numerical integration of stiff or high-order dynamics may exceed the available compute budget at high control frequencies (e.g., $\geq 1\,\text{kHz}$), and solver-related memory buffers can dominate the footprint. FMI 2.0 itself provides no native scheduling interface, leaving timing integration and determinism to the host environment. As a result, timing behavior depends on the integration pattern (polling loop versus scheduled task), the chosen master/orchestration policy, and whether the FMU performs dynamic memory allocation. This can introduce jitter unless FMUs are carefully profiled, configured, or adapted for the target execution context [9, 20]. Moreover, memory constraints limit the sizes of internal state, parameter maps, and solver work vectors; these constraints are further amplified when multiple FMUs must be executed concurrently or when communication occurs over bandwidth-limited links [16]. Therefore, embedded execution feasibility is not only a function of model fidelity, but of step-size requirements, execution-time variability, and memory allocation patterns.

FMI 3.0 addresses part of this gap by introducing clocks and a dedicated Scheduled Execution interface, providing semantic building blocks to align model computations with externally managed scheduling (e.g., OS/RTOS-level scheduling) [23, 24, 45]. Scheduled Execution provides

explicit partitioning and activation semantics, but it does not, by itself, guarantee hard real-time correctness: WCET characterization, bounded memory, and tool-supported schedulability analysis remain external requirements. However, practical deployment still depends on the maturity of the toolchain and on the availability of lightweight runtimes and integration patterns that preserve determinism, bounded memory, and analyzability [33, 65].

These constraints motivate alternative strategies such as code generation, solver restructuring, model partitioning, and hybrid architectures where high-fidelity models remain in co-simulation while reduced or deployment-oriented variants execute on-device [12, 25]. Accordingly, the literature splits into two families: runtime-based FMU execution on embedded/edge platforms for experimentation and validation, and code-generation-oriented pipelines that restrict semantics to enable analyzable, certifiable execution.

Across the reviewed work, embedded use of FMI-based artifacts recurs in a small set of deployment archetypes that differ in their execution contracts and evidence strength:

- **A1: Embedded FMU as an on-device reference/oracle (validation continuity).** FMUs are executed in SIL/HIL pipelines to validate controllers; real-time deadlines are not necessarily hard constraints, but repeatability and traceability are central [17, 59].
- **A2: Embedded FMU for monitoring/estimation/diagnostics (soft real-time).** Lightweight execution supports local analytics at moderate rates; feasibility depends on bounded event density and limited stiffness [12].
- **A3: Real-time-capable FMU execution on edge nodes (controller-in-the-loop).** FMUs run near the controller with bounded stepping policies; end-to-end latency and jitter become dominant.
- **A4: Deployable embedded control functions via restricted semantics (certification-oriented).** Code generation targets bounded memory and timing transparency (eFMI/GALEC) and supports safety arguments [25, 26].

These archetypes clarify why FMU runs on device is not a sufficient claim: the required execution contracts differ substantially between validation continuity (A1) and safety-relevant hard real-time control (A4).

To compare embedded feasibility across heterogeneous reports, evidence should be grounded in measurable execution artifacts. Minimal reporting elements include: sampling period and deadline (or target loop rate), measured step-time distribution (mean and tail behavior), jitter envelope under representative load, and peak memory/stack usage. Where such measurements are absent, feasibility claims are treated as inferred rather than demonstrated, consistent with the quality assessment in Sect. 2. Despite these constraints, proof-of-concept and industrial experiments demonstrate that FMI-based artifacts can be executed on embedded and edge platforms when the model class and execution objectives are appropriately constrained.

A representative example is the `ufmu` framework by Reynaud et al. [12], which enables execution of FMI 2.0-compliant FMUs within a MicroPython environment targeting microcontrollers such as the ESP32. In this approach, FMU source code is integrated into firmware, and FMI APIs are exposed via scripting, enabling rapid prototyping and reproducible experiments on resource-constrained devices. The reported results indicate that non-stiff models can be stepped efficiently at moderate sampling rates, while footprint and solver restrictions remain practical limitations for high-fidelity dynamics [12] highlighting that lightweight runtimes are feasible primarily for models with limited stiffness and bounded computational variability.

Complementary approaches embed FMUs directly in C/C++ firmware or ECU-oriented environments with static linkage and platform-specific integration. Mikelsons and Samlaus [17] report FMI-based validation setups for embedded control units in industrial contexts, illustrating how FMU-based reference models can support repeatable verification and traceability across SIL/HIL pipelines, even when direct hard real-time deployment of full-fidelity models is not the objective. This category is important for actuator engineering because it operationalizes interoperability as validation continuity. The FMU acts as a high-fidelity reference (oracle) for testing embedded controllers under realistic scenarios.

Additional integration options are emerging in robotics and automation platforms. For example, FMI Terminal Block bridges FMI models to IEC 61499 function blocks [47], and ROS 2 integration layers enable FMUs to participate in controller-in-the-loop experiments within robotic middleware stacks [48, 52]. Such deployments enable actuator-level estimation, diagnostics, and adaptation to be colocated with hardware, aligned with modern edge computing architectures. Nonetheless, runtime overhead, integration complexity, and limitations on determinism motivate the adoption of more structured, deployment-oriented solutions for safety-critical actuator control. In particular, middleware-induced latency and jitter can invalidate assumptions made during desktop co-simulation unless timing semantics are explicitly modeled and enforced.

**Table 12** Typical execution-contract coverage of embedded FMI deployment approaches (qualitative)

| Approach family | Timing contract | WCET evidence | Memory boundedness | Best fit |
|---|---|---|---|---|
| Runtime FMU on MCU/edge | Medium (host-dependent) | Low–Medium | Medium | A2/A3 (monitoring, prototyping) |
| FMU as reference/oracle in validation | Medium (bench-driven) | Medium (bench-specific) | Medium | A1 (SIL/HIL continuity) |
| FMI 3.0 SE on RTOS | Medium–High (semantic hooks) | Medium (toolchain-dependent) | Medium | A3 (timing-aware integration) |
| eFMI/GALEC code generation | High (restricted semantics) | High (analysis-friendly) | High | A4 (safety-relevant deployment) |

**Table 13** Embedded FMI approaches for industrial applications: runtime-based execution versus code-generation-based deployment

| Approach | Representative work | Reported timing/resource evidence | Comparative interpretation | Claim supported |
|---|---|---|---|---|
| Lightweight FMU runtime on MCU/edge | Reynaud et al. (uFMU, MicroPython+ESP32) [12] | Step-time and memory-footprint evidence for a lightweight, non-stiff MCU prototype; jitter and WCET NR | Strongest direct evidence for constrained runtime feasibility, but platform/model dependent. | Feasible for prototyping, monitoring, and non-safety-critical local functions. |
| FMU reference models in ECU validation | Mikelsons & Samlaus [17] | HIL/ECU validation setting reported; complete embedded deployment tuple NR | Supports validation continuity rather than on-device execution of full-fidelity FMUs. | Credible for SIL/HIL reference-model use. |
| eFMI / GALEC code generation | Lenord et al. [25] | Bounded-memory and WCET-oriented design rationale; cross-method numeric benchmark NR | Best evidence for analyzability-by-construction, but not a measured jitter comparison. | Appropriate for certification-oriented deployment arguments. |
| eFMI to AUTOSAR ECU deployment | Armugham et al. [26] | Industrial ECU/AUTOSAR integration evidence; complete jitter/WCET/memory tuple NR | Demonstrates industrial deployment path, but evidence remains case-study specific. | Supports safety-relevant ECU integration under restricted workflows. |
| Model-based translation to PLC execution | Walther et al. [49] | PLC execution semantics reported; high-bandwidth actuator timing metrics NR | Relevant for industrial control deployment, but weaker for fast actuator dynamics. | Useful for PLC-oriented control logic migration. |

## 7.2 eFMI and Code Generation for Embedded Actuator Control

To address limitations of standard FMI runtimes, the eFMI specifies a deployment-oriented framework for transforming validated physical models into deterministic, analyzable embedded code [25]. Instead of executing a general FMU runtime on-device, eFMI introduces an intermediate representation, GALEC, designed to bridge equation-based modeling and embedded software production workflows [25, 66, 67]. This shift replaces the interpretation of a general simulation artifact with a restricted semantics that can be compiled and analyzed.

This workflow promotes fixed memory layouts, restricted execution semantics, and tool-driven analysis capabilities (e.g., bounded memory usage and worst-case execution time reasoning), which are prerequisites for certified embedded development processes in safety-relevant domains [18, 19, 27]. In practice, eFMI targets execution-level interoperability: it aims to preserve traceability from validated models while enforcing constraints required by schedulability analysis and certification processes. In the eFMI–AUTOSAR case study reported by Armugham et al., Modelica-based actuator control functions are transformed via GALEC into AUTOSAR-compliant software components and deployed to production-grade ECU stacks, illustrating feasibility for deterministic embedded integration under industrial constraints [25, 26].

These approaches represent a shift from simulation-time model reuse toward deployable, certifiable model-driven software generation, maintaining interoperability while meeting embedded execution constraints. However, they typically require restricting model classes, numerical methods, and language features to enable bounded execution, which introduces an explicit fidelity–analyzability trade-off that must be managed at design time. From a computational perspective, the key advantage is that these restrictions enable stronger claims: bounded memory, predictable control flow, and a clearer pathway to timing evidence and qualification artifacts. To clarify what each embedded approach covers regarding execution contracts, Table 12 summarizes the typical levels of timing, resource, and semantic control. This complements Table 13 by focusing on contract coverage rather than on qualitative pros/cons.

To structure the embedded evidence, Table 13 now contrasts representative deployment paths according to reported

timing/resource evidence and the claims that can be supported from that evidence. This distinction is important because only a subset of the reviewed works reports execution-time or memory information, and none reports a complete jitter/WCET/memory comparison across methods.

From an embedded-systems perspective, the execution of FMI models enables continuity across design, validation, and deployment. Runtime-based approaches preserve FMI compatibility for on-device experimentation and diagnostics, while eFMI/GALEC-based code generation provides a pathway to deterministic and certification-aligned embedded actuator control. Together, these approaches extend FMI interoperability toward the physical edge of industrial actuator systems, but they also make explicit that execution-level interoperability requires enforceable timing/resource contracts beyond FMU packaging alone.

## 8 FMI Tooling Ecosystem and Industrial Readiness

The adoption of FMI as an interoperability standard in industrial applications is strongly influenced by the maturity and availability of supporting tools. While the FMI specification defines the semantic and interface foundations for model exchange and co-simulation, its practical impact depends on the extent to which modeling environments, co-simulation orchestrators, runtimes/SDKs, and deployment toolchains implement these capabilities consistently [20, 23, 35]. This section reviews the FMI tooling ecosystem and assesses its readiness for high-fidelity simulation and actuator control in embedded industrial systems, with emphasis on (i) feature coverage across FMI versions, (ii) validation and conformance mechanisms, and (iii) deployment-oriented workflows. To avoid purely qualitative claims, industrial readiness is evaluated relative to execution-level requirements that dominate actuator-centric embedded control: timing determinism, bounded resources, and verifiable integration behavior.

### 8.1 Overview of Available FMI Tools

The FMI ecosystem spans multiple layers that map to different phases of the engineering lifecycle:

1. **Authoring and export tools.** Equation-based modeling environments such as OpenModelica [44] and Dymola [36], together with multi-domain simulation suites (e.g., MapleSim, SimulationX), provide native support for exporting high-fidelity models as FMUs. These tools are commonly used to create validated plant/actuator models that are later reused in virtual commissioning and system integration [35, 38, 39], typically prioritizing modeling expressiveness and solver richness over embedded resource constraints.
2. **Desktop simulation and co-simulation environments.** Industrial simulation platforms and integration toolchains support importing FMUs and composing multi-domain systems. In actuator-centric workflows, this layer is typically where control logic, plant dynamics, and test benches are combined for MiL/SiL evaluation, and where model ownership/IP boundaries are enforced through black-box FMUs [20]. At this level, interoperability is largely model-level: success is measured by import/export compatibility and numerical plausibility rather than by schedulability or bounded execution time.
3. **Orchestration frameworks and runtime libraries.** Beyond integrated desktop tools, FMI adoption depends on standalone runtimes and SDKs that embed FMUs into custom applications, test rigs, and digital twin services. Examples include the open-source FMI Library (FMIL) [68] and Python toolchains such as FMPy [69] and PyFMI [70]. These runtimes differ in FMI-version support, ME/CS coverage, performance characteristics, and integration complexity. For actuator-centric use, the critical differentiator is whether the runtime exposes controllable stepping/synchronization policies and whether it can be integrated without introducing unbounded latency or memory allocation behavior.
4. **Robotics and middleware integration.** Tooling such as `gazebo-fmi` [52] and ROS 2 FMI adapters [48] connects FMUs to cyber-physical experimentation stacks, enabling controller-in-the-loop testing and actuator-dynamics injection into ROS/Gazebo pipelines [30, 31]. This layer is particularly relevant for actuator validation, where timing behavior and sensor/actuator interfaces must be exercised under realistic closed-loop conditions. However, middleware-induced latency and jitter can dominate end-to-end behavior, so success in this layer must be evaluated in terms of timing consistency in addition to interface connectivity.
5. **Validation and conformance infrastructure.** Industrial readiness also depends on validation tooling that improves reproducibility and reduces integration risk. The FMI community provides validation guidance and tool references (e.g., FMU validation workflows based on FMPy) [71]. Additionally, cross-tool interoperability initiatives such as the FMI Cross-Check process and its published rules/results provide an ecosystem-level mechanism to surface incompatibilities and improve tool robustness [72, 73]. In practice, conformance and cross-checking are essential for industrial adoption because they mitigate vendor-specific interpretations of FMI semantics and address corner cases.

Overall, FMI 2.0 remains broadly supported across commercial and open-source tools, while FMI 3.0 feature adoption is still emerging across the ecosystem [23, 33, 35]. In the PRISMA corpus, this statement refers to evidence status rather than to a simple tool-count claim: only two studies directly discuss FMI 3.0 clocks/SE, both conceptually, and no actuator-centric embedded study empirically evaluates FMI 3.0 clocks/SE with reported jitter, WCET, and memory footprint. Embedded and real-time deployment paths therefore rely on specialized runtimes (e.g., `ufmu`) [12] and/or deployment-oriented code-generation workflows such as eFMI [25]. Open-source contributions such as FMUiL [31] and RL training frameworks built on FMUs [60] further extend the ecosystem into in-the-loop testing and data-driven control policy development. Consequently, the ecosystem currently exhibits a bifurcation: mature tooling for ME/CS (FMI 2.0), and emerging tooling for timing-aware or certifiable execution (FMI 3.0 SE and eFMI/GALEC).

### 8.2 Gaps Between FMI Standards and Tool Support

Despite the conceptual clarity of FMI, gaps remain between specification capabilities and industrial tooling maturity, particularly for embedded actuator control. FMI 3.0 defines timing semantics and Scheduled Execution at the specification level [23, 33]. However, the reviewed empirical actuator literature does not yet show end-to-end FMI 3.0 SE deployment on embedded targets with quantified timing/resource evidence. Tool implementations and reference patterns for embedded real-time execution are therefore not yet uniformly available in mainstream industrial toolchains [35]. A key limitation is that a mixed FMI 2.0–3.0 compatibility label does not necessarily mean that clocks, Scheduled Execution, or FMI 3.0 timing features were tested.

Furthermore, many runtimes target desktop environments and assume abundant resources. Execution on constrained devices demands lightweight implementations, static linking, careful memory strategies, and tight integration with real-time schedulers. These requirements are only partially addressed by general-purpose FMI runtimes, motivating research-driven solutions and hybrid deployments where embedded nodes execute reduced or code-generated variants [12, 25]. For actuator-centric embedded control, the dominant gap is execution-level interoperability: most runtimes do not provide bounded-memory guarantees, do not expose WCET-relevant execution profiles, and are not designed for schedulability analysis under RTOS constraints.

Certification and safety considerations amplify these limitations. Standard FMI runtimes are not designed to provide analyzability guarantees (e.g., bounded memory usage and WCET arguments) required by safety standards such as ISO 26262 or IEC 61508. In contrast, eFMI addresses these constraints by generating analyzable C code from restricted models and supporting integration into certification-oriented embedded workflows [18, 19, 25, 26], effectively shifting the toolchain from FMU execution to model-to-code transformation under restricted semantics.

### 8.3 Operational Readiness Levels for FMI Toolchains

To make industrial readiness actionable, we summarize typical toolchain maturity in four qualitative readiness levels defined by execution-level capabilities. These levels are not vendor-specific; they describe what a workflow can credibly guarantee for actuator-centric embedded control:

- **R0 (Exchange-ready):** FMU import/export works reliably for desktop MiL/SiL. Determinism is not claimed; solver/step policies are largely implicit.
- **R1 (Integration-ready):** The toolchain exposes stepping/orchestration controls (communication step bounds, solver disclosure) and supports reproducible co-simulation configurations for timing-sensitive validation.
- **R2 (Timing-aware):** FMI 3.0 clock semantics and/or Scheduled Execution workflows are supported in a way that can be mapped to external scheduling; profiling hooks enable jitter/latency characterization under load.
- **R3 (Certification-oriented):** The workflow provides a credible pathway to bounded-memory, WCET-friendly execution artifacts (e.g., eFMI/GALEC or equivalent restricted code generation) and supports evidence generation aligned with safety processes.

These levels clarify why FMI 2.0 ecosystems are often strong at R0/R1, while embedded actuator control requires at least R2 (timing-aware) and, for safety-relevant deployment, R3 (certification-oriented).

### 8.4 Timing-Semantics Expectations for FMI 3.0 Tool Support

Because FMI 3.0 support is frequently reported as a binary capability, Table 14 lists minimal expectations for timing-aware tooling that claims support for clocks and Scheduled Execution in actuator-centric workflows. For the S1-A studies labeled FMI 2.0–3.0, the label is interpreted conservatively: unless clocks or Scheduled Execution are explicitly evaluated, the study is counted as compatibility/tooling evidence rather than empirical FMI 3.0 timing-semantics evidence.

**Table 14** Minimal expectations for timing-aware FMI 3.0 tool support in actuator-centric workflows (conceptual checklist)

| Capability | What the toolchain should expose | Why it matters |
|---|---|---|
| Clock semantics | Clock types, activation conditions, clocked-variable updates | Makes sampled-data assumptions explicit. |
| SE partition mapping | Partition identifiers and activation points | Enables RTOS task mapping and dependency reasoning. |
| Stepping / synchronization control | Bounded step policy; explicit communication points | Reduces host-dependent trajectories and coupling artifacts. |
| Profiling hooks | Step-time logging; latency/jitter monitoring under load | Turns feasibility into measurable execution evidence. |
| Conformance evidence | Cross-check or validation results for timing features | Reduces vendor-specific interpretation risks and corner cases. |

## 8.5 Implications for Industrial Adoption

The FMI tooling ecosystem is in transition. FMI 2.0 is robust and widely used for model exchange and co-simulation, whereas FMI 3.0 introduces timing features whose actuator-specific embedded evidence remains limited [23, 33]. Consequently, industrial users aiming at embedded actuator control typically combine FMI 2.0-based validation/co-simulation with deployment-oriented mechanisms when determinism or compliance is required. eFMI/AUTOSAR evidence supports the code-generation path [26], while model-to-PLC translation illustrates a domain-specific industrial deployment path [49]. This mixed strategy reflects an engineering reality: the same actuator model may be used as a high-fidelity reference FMU during verification, while a constrained executable artifact (generated or reduced) is deployed on the embedded target.

Industrial readiness can be operationalized by three toolchain-level questions:

- **Q1:** Does the workflow provide explicit stepping/scheduling semantics compatible with real-time execution?
- **Q2:** Can it enforce bounded resource usage (memory and allocation behavior)?
- **Q3:** Does it provide validation evidence (conformance tests, cross-check results, or reproducible benchmarks) for timing-critical scenarios?

In conclusion, FMI provides a strong foundation for interoperability, but its practical utility for embedded actuator control depends critically on the surrounding toolchain: orchestration maturity, runtime performance, validation infrastructure, and certification-friendly deployment workflows.

Table 15 provides an ecosystem-level view of FMI-supporting tool categories, indicating FMI-version coverage (2.0 vs 3.0), execution-mode support (ME/CS/SE),

**Table 15** Overview of the FMI tooling ecosystem and its readiness for high-fidelity simulation and embedded actuator control

| Tool category | Examples / evidence source | FMI 3.0 / SE evidence status | Reported timing/resource metrics | Readiness interpretation for actuator control |
|---|---|---|---|---|
| Modeling and authoring tools | OpenModelica [44], Dymola [36], SystemModeler | Tool support is partial/tool-dependent; S1-A provides no embedded SE benchmark | NR for jitter/WCET/memory | Strong R0/R1 model-generation role; embedded readiness depends on downstream runtime or code generation. |
| Desktop simulation and co-simulation frameworks | Simulink, SimulationX [39], FMI co-sim orchestrators | FMI 3.0 compatibility does not imply tested clock/SE behavior | Host/tool timing settings may be configured, but complete embedded tuple NR | Useful for MiL/SiL and virtual commissioning; weak evidence for hard real-time deployment. |
| FMI SDKs and runtime libraries | FMILibrary [74], FMPy [69], PyFMI [70] | Mostly FMI 2.0/ME/CS evidence in actuator corpus; FMI 3.0 SE evidence limited | NR for actuator embedded jitter/WCET/memory | R1 integration support; schedulability must be supplied by host application. |
| Robotics integration frameworks | `gazebo-fmi` [52], ROS2 FMI Adapter [48], MoRC [30] | Predominantly FMI 2.0 CS/controller-in-the-loop evidence | Loop timing discussed in prototypes; complete tuple NR | R1/R2-adjacent for controller-in-the-loop validation; middleware jitter remains external. |
| Embedded / edge runtimes | `ufmu` [12], statically linked FMUs | Restricted FMI 2.0 runtime evidence; not FMI 3.0 SE | Step-time and memory footprint reported for constrained prototype; jitter/WCET NR | Best direct runtime evidence for non-safety-critical MCU/edge use, with model restrictions. |
| Code-generation toolchains | eFMI / GALEC [25], AUTOSAR case study [26] | Separate from FMI 3.0 SE runtime evidence | Bounded-memory/WCET-oriented claims; cross-method measured tuple NR | R3 pathway for safety-relevant deployment through restricted, analyzable artifacts. |
| Digital twin and industrial integration | FMI+OPC UA, Twin Builder, lifecycle workflows | Mostly conceptual or integration-level mixed-version evidence | NR for actuator embedded timing/resource metrics | Supports lifecycle interoperability, but not direct execution-level guarantees. |
| Validation and conformance tooling | FMI validation workflows [71], FMI Cross-Check [72, 73] | Conformance evidence is valuable but not actuator-specific timing validation | NR for embedded actuator jitter/WCET/memory | Reduces import/export risk; timing-aware conformance suites remain a research gap. |

**Table 16** Comparison of FMI-based deployment approaches for actuator-centric embedded control

| Approach | Execution Semantics | Determinism | WCET / Timing Analysis | Typical Actuator-Centric Use Case |
|---|---|---|---|---|
| Desktop FMI (ME / CS) | Tool-driven execution; solver-managed integration and event handling | Low | Not analyzable | Model development; system simulation; virtual commissioning; design-space exploration. |
| Restricted FMI runtime on edge / MCU | Step-based execution; static linking; limited scheduling control | Medium | Partially analyzable (platform and implementation dependent) | Edge diagnostics; state estimation; monitoring; non-safety critical local control. |
| FMI 3.0 with Scheduled Execution | Clocked execution; explicit activation points and scheduling semantics | Medium–High | Potentially analyzable (depends on tool support and WCET characterization) | Real-time integration; early embedded prototyping; timing-aware co-simulation. |
| eFMI / GALEC code generation | Static, bounded, analyzable C code derived from restricted models | High | Certification oriented analyzability (workflow and toolchain dependent) | Safety-critical actuator control; hard real-time embedded deployment. |

**Table 17** Research agenda for execution-level interoperability of FMI-based actuator workloads: problems, evidence, and near-term milestones

| Problem | Why it is hard | What evidence counts | Near-term milestone |
|---|---|---|---|
| Timing contracts across FMU boundary | FMI packaging does not encode deadlines, jitter, or WCET; stepping remains host-dependent | Public benchmarks with jitter/WCET under load; disclosed activation assumptions | Embedded profile metadata + reference runtime binding. |
| Compositional schedulability | Multi-FMU interference; shared I/O; priority inversion; coupling delays | Schedulability studies with realistic interference; reproducible task sets | Reference case study: multi-FMU actuator loop on RTOS. |
| Solver determinism and reproducibility | Different solvers/steps yield different trajectories; event ordering differs across tools | Solver/step disclosure + regression tests; bounded error envelopes | Numerical-disclosure conventions + conformance tests. |
| Fidelity-preserving reduction | Must preserve control-relevant behavior under FMI interfaces | Behavioral equivalence tests (frequency/delay margins, invariants) | Multi-fidelity tiers linked by automated tests. |
| Certification-ready pipelines | Requires evidence artifacts, tool qualification, coding standards | End-to-end safety case + timing/memory reports | Qualification-style demonstrator for one actuator component. |
| Secure execution without RT regressions | Security mechanisms may introduce latency/jitter and memory overhead | Measured overhead bounds; deterministic sandboxing patterns | Signed FMU + bounded-latency execution wrapper. |

and practical readiness for embedded actuator-oriented workflows. The matrix now separates tool-support claims from reviewed quantitative evidence; NR denotes that jitter, WCET, or memory-footprint data were not reported in the reviewed actuator-relevant evidence.

## 9 Open Challenges and Research Directions

Despite significant progress in standardization and tooling, the use of FMI for high-fidelity actuator simulation and embedded control remains constrained by unresolved challenges spanning execution semantics, deployability, assurance, and ecosystem maturity. Importantly, many limitations are not specification-level issues but end-to-end workflow issues: from modeling assumptions and solver behavior to runtime integration, certification evidence, and long-term maintainability. In actuator-centric systems, these challenges can be organized around execution-level interoperability: the ability to preserve validated behavior under bounded resources and predictable timing. Table 16 summarizes the practical implications of different FMI-based deployment paths.

To make the research directions actionable (and evidence-weighted), Table 17 summarizes a compact agenda of high-impact problems, why they are hard, what evidence constitutes progress, and near-term milestones that would materially advance execution-level interoperability.

### 9.1 Determinism, Scheduling, and Real-Time Guarantees

Hard real-time actuator control requires bounded latency, bounded jitter, and predictable resource usage under an RTOS scheduler. FMI 3.0 introduces clocks and Scheduled Execution as first-class interface concepts [23, 24, 33]. However, alignment with real-time scheduling still has to be demonstrated through scheduler bindings, task models, and measured timing evidence rather than inferred from the specification alone. Practical deployment therefore still faces three gaps:

- **Timing contracts across the FMU boundary**: FMI does not yet provide a widely adopted way to express and validate timing contracts (e.g., maximum step time,

scheduling constraints, partition dependencies) that can be checked by tooling.
- **Compositional schedulability analysis**: multi-FMU systems require analysis beyond individual components (e.g., interference, priority inversion, shared communication and I/O), which is largely absent in current FMI toolchains.
- **Runtime integration patterns**: mapping FMI execution to RTOS task models (AUTOSAR OS or FreeRTOS-like architectures) is still mostly done ad hoc, and reproducible reference architectures for actuator loops remain scarce.

A concrete research direction is to define industrial FMI embedded profiles that pair FMI 3.0 Scheduled Execution with explicit schedulability metadata (periods, deadlines, partition dependencies, and resource budgets), together with reference master algorithms and scheduler bindings. To be actionable, such profiles should be accompanied by benchmark suites that report WCET distributions, jitter envelopes under realistic interference loads, and closed-loop stability sensitivity to timing perturbations. Such profiles would be particularly valuable for actuator loops operating at high frequencies (e.g., hundreds of Hz to kHz), where solver choice, event handling, and data exchange overhead dominate timing behavior. A pragmatic intermediate step is to standardize measurement protocols (how jitter/WCET are collected and under what load), since inconsistent measurement practices currently prevent fair comparison of embedded feasibility claims.

### 9.2 Model Complexity, Solver Semantics, and Fidelity Management

High-fidelity actuator FMUs often embed nonlinearities (friction, backlash, saturation), multi-domain couplings (electro-thermal-mechanical), and switching behaviors. These features increase stiffness and event density, stressing both real-time execution and numerical robustness. On embedded targets, this leads to unavoidable trade-offs between fidelity and feasibility [16]. Two under-addressed issues in current FMI-centric workflows are:

- **Fidelity-preserving reduction under interface constraints**: model reduction or surrogate substitution must preserve the input-output behavior relevant for control, while remaining FMI-compliant and traceable to the original model.
- **Solver determinism and reproducibility**: even with the same FMU, different host solvers or step-size strategies can yield different trajectories, complicating validation, regression testing, and certification evidence.

A promising direction is to formalize multi-fidelity equivalence classes for actuator FMUs: a full model (design/reference), a reduced model (real-time capable), and a deployment artifact (embedded/verified), linked by traceability artifacts and automated behavioral tests. These tests should target actuator-relevant invariants (e.g., passivity/energy bounds for interaction tasks, saturation and thermal-limit behaviors, delay margins, and frequency-response envelopes), rather than only pointwise trajectory matching. This also encourages standardized reporting of numerical settings (event tolerances, step size limits) and automated compliance testing targeting behaviors relevant to actuators. In addition, reproducibility would benefit from numerical disclosure conventions: explicit solver families, tolerance settings, and event-handling policies recorded as part of the FMU execution metadata. A practical deliverable is a reference actuator benchmark suite comprising stiff, event-rich FMUs with known invariants, enabling regression testing across tools and orchestrators.

### 9.3 Certification Evidence and Qualification-Ready Toolchains

Many embedded actuator controllers fall under safety standards (e.g., ISO 26262, IEC 61508) and, in aerospace contexts, DO-178C/ED-12C [75]. These processes require analyzable artifacts, traceable requirements, and evidence that timing and memory bounds are satisfied.

Standard FMI runtimes are primarily designed for general-purpose simulation and typically do not provide the assurance hooks needed for certification-grade workflows. In contrast, eFMI pursues a restricted, analyzable path via code generation (GALEC and related artifacts), enabling predictable memory layout and timing-analysis arguments [25, 76, 77]. The AUTOSAR case-study evidence is treated as industrial deployment evidence for this restricted code-generation path [26]. However, broad adoption requires:

- **Tool qualification arguments** and repeatable evidence generation.
- **Coding-standard alignment** (e.g., MISRA-oriented constraints for generated C code) [78].
- **Traceability mapping** from model elements to safety cases and verification results.

Research opportunities include integrating eFMI-style analyzability with FMI 3.0 Scheduled Execution semantics, and defining qualification-ready pipelines that automatically generate the documentation artifacts expected by assessors (requirements trace matrices, configuration baselines, test evidence, and timing/memory reports). A practical milestone would be reference case studies that demonstrate end-to-end safety arguments for actuator components, including WCET measurement methodologies and regression test suites linked to model revisions. To strengthen industrial credibility, such case studies should explicitly document the model restrictions that enable analyzability (e.g., bounded loops, bounded events, no dynamic allocation), making the fidelity assurance trade-off transparent.

### 9.4 Secure Exchange, IP Protection, and Trustworthy Execution

In industrial supply chains, FMI is frequently used to exchange black-box models. While FMI packaging provides practical obscurity, it does not define a standardized security or IP-protection layer, and risks remain (reverse engineering of binaries, parameter extraction, malicious payloads, and untrusted model execution) [79, 80]. Consequently, IP concerns directly limit adoption in actuator and drive domains where models encode proprietary know-how.

Emerging research has begun to address the secure exchange and execution of FMI-based models in industrial contexts [79, 81]. A relevant roadmap here includes:

- **Trusted packaging and provenance**: signing, integrity verification, and supply-chain traceability for FMUs.
- **Sandboxed execution**: containment of FMUs (e.g., hardened runtimes or platform-isolation approaches) to reduce operational risk.
- **Licensing and access control**: standardized mechanisms for runtime authorization without breaking interoperability.

For embedded actuator deployment, security requirements are closely coupled with real-time requirements: sandboxing and verification mechanisms must be designed so they do not introduce unbounded latency, uncontrolled memory growth, or non-deterministic execution effects. Accordingly, secure FMI execution should be evaluated with the same execution-contract lens used for determinism (latency/jitter bounds, bounded memory, and predictable control flow).

### 9.5 Validation Infrastructure and Actuator-Focused Benchmarking

Finally, industrial adoption depends on validation infrastructure and predictable tool behavior. FMI cross-checking and validation tools exist, but integrated, actuator-focused compliance testing remains limited and fragmented. The FMI Cross-Check rules and related validation initiatives highlight the importance of systematic interoperability testing [72, 82], and community tooling (e.g., compliance checkers) demonstrates progress [83]. However, FMI 3.0 feature support (clocks, SE) still varies across vendors and open-source ecosystems. A key research and engineering priority is to develop:

- **Embedded-oriented conformance suites** (timing, memory, event density, actuator benchmarks).
- **Reference FMUs** for actuator dynamics (stiffness, friction, saturation, delays).
- **Public, reproducible benchmarking** of runtimes and toolchains under embedded constraints.

To maximize impact, such benchmarks should report not only performance metrics (Hz, latency, jitter, footprint) but also control-relevant outcomes (tracking error under timing perturbations, stability margins, and failure modes under overload). A near-term deliverable is a minimum viable conformance suite for FMI 3.0 clocks/SE that includes actuator-style event density and stiffness, since generic FMU tests rarely stress the timing failure modes that matter in closed-loop control.

Despite its maturity in tool interoperability, actuator-centric embedded control still demands advances in real-time integration, fidelity management, certification evidence, secure distribution, and ecosystem-wide validation. Addressing these gaps is essential to transition FMI from successful desktop co-simulation to dependable infrastructure at the embedded actuator level.

## 10 Conclusions

This review has examined the Functional Mock-up Interface (FMI) as an interoperability backbone for high-fidelity simulation and actuator-centric control workflows targeting industrial embedded systems. By explicitly placing actuator dynamics (multi-domain, nonlinear, timing-sensitive) and

embedded execution constraints (bounded resources, deterministic timing, industrial compliance) at the center of the analysis, the review consolidates the current state of the art and clarifies the technical barriers that still limit seamless continuity from model-based design to embedded deployment. Across the surveyed literature, the central message is that interoperability for actuators is fundamentally execution-sensitive: model exchange alone is insufficient unless stepping, timing, and resource semantics are made explicit and testable.

The review contributes an execution-focused comparison of FMI 2.0 versus FMI 3.0 semantics (clocks and Scheduled Execution), a synthesis of actuator fidelity requirements mapped to numerical and timing consequences, and an evidence-weighted assessment of embedded deployment trajectories ranging from runtime-constrained FMU execution to analyzable eFMI/GALEC code generation.

From an actuator-engineering perspective, the implications of FMI extend beyond tool interoperability. Actuators are inherently multi-domain systems in which electrical, mechanical, thermal, and control dynamics interact under tight timing and safety constraints. Ensuring that high-fidelity models of friction, saturation, backlash, compliance, and drive electronics remain consistent across simulation, validation, and deployment phases is critical for performance, stability margins, and lifecycle efficiency. By treating actuator models as portable, interface-stable artifacts, FMI enables continuity not only at the software level but also in the representation of physical behavior. This continuity is particularly valuable in applications where controller tuning, fault diagnostics, energy optimization, and safety validation depend on consistent cross-phase actuator dynamics. However, for actuator loops, continuity must be evaluated not only by interface compatibility but by preservation of closed-loop behavior under the timing and solver semantics imposed by the target execution environment.

FMI 2.0 remains the most mature and widely adopted baseline for model exchange and co-simulation, enabling modular reuse of validated actuator and drive models across heterogeneous tools and organizational boundaries. In industrial practice, FMI 2.0 has proven particularly effective for virtual integration, supplier model exchange, and validation workflows such as SIL and HIL, where black-box encapsulation and IP protection are key enablers. However, as actuator models move closer to embedded closed-loop execution, structural limitations become evident: the absence of explicit scheduling semantics, limited timing guarantees, and reliance on host-specific execution policies complicate deterministic integration under RTOS and safety-oriented constraints. In other words, FMI 2.0 largely guarantees model-level interoperability, while execution-level interoperability remains external and implementation-dependent.

FMI 3.0 represents a significant semantic step forward by introducing clocks and Scheduled Execution, which directly address discrete-time coordination and scheduler-aligned execution capabilities that are essential for real-time actuator control. Conceptually, these extensions narrow the gap between simulation-oriented interfaces and embedded control architectures. However, this review highlights a persistent maturity gap: tool and runtime support for FMI 3.0 features remains uneven, and robust patterns for hard real-time execution, verification, and industrial qualification are still emerging. As a result, adoption in production-grade actuator systems is currently constrained by ecosystem readiness rather than by specification intent alone. Therefore, FMI 3.0 should be viewed as an enabler of timing-aware interoperability, whose practical value depends on the availability of analyzable runtimes, schedulability-aware integration patterns, and conformance evidence.

Regarding embedded deployment, two complementary trajectories emerge from the literature. On one hand, lightweight runtimes and wrappers demonstrate the feasibility of executing FMUs on constrained platforms and edge devices, supporting rapid prototyping, localized model reuse, and early-stage validation. On the other hand, code-generation-oriented approaches such as eFMI and GALEC explicitly target determinism, analyzability, and certification-relevant constraints by producing embedded code with bounded resource usage and well-defined execution semantics. Rather than being competing alternatives, these approaches form a continuity: runtime-based execution is well-suited for exploratory development and validation, while eFMI-style compilation is better-suited for industrial-grade integrated deployment, where time guarantees and regulatory compliance are important. This continuum operationalizes the embedded execution gap: it separates workflows that prioritize model reuse from workflows that prioritize enforceable execution contracts.

Across the reviewed work, four cross-cutting challenges emerge as decisive for establishing FMI as a dependable interoperability layer for industrial actuator control in embedded settings. First, end-to-end determinism and schedulability integration must be strengthened through embedded-focused conformance profiles and reference execution

models. Second, fidelity management and systematic model reduction workflows are needed to preserve control-relevant behavior while respecting embedded resource constraints. Third, certification and tool-qualification pathways must be clarified to connect FMI and eFMI artifacts with industrial safety and compliance processes. Fourth, standardized mechanisms for secure model distribution and intellectual property protection beyond basic black-box packaging are required to support large-scale industrial adoption. For actuator-centric systems, FMI should be understood not as a runtime technology, but as an interoperability contract whose deployment semantics must be explicitly engineered. A practical implication is that future FMI-ready actuator workflows should include: explicit timing metadata and scheduler bindings, reproducible numerical configurations and solver disclosure, benchmarkable latency/jitter/footprint reports, and traceability artifacts linking design FMUs to deployment code.

For practitioners aiming to use FMI in actuator-centric embedded workflows, the following actions materially improve credibility and reproducibility:

- **Disclose execution assumptions:** solver family, step-size bounds (or communication-step policy), and event-handling/tolerance settings.
- **Treat timing as a first-class artifact:** report sampling periods, deadlines, and measured jitter/latency under representative load.
- **Separate reference from deployment:** use high-fidelity FMUs as validation oracles (SIL/HIL) and deploy constrained artifacts (reduced or code-generated) when bounded execution is required.
- **Adopt contract-based validation:** regression tests should target actuator invariants (saturation/thermal limits, delay margins, passivity/energy bounds) rather than only trajectory snapshots.
- **Prefer qualification-oriented paths for safety:** when certification is relevant, favor analyzable pipelines (eFMI/GALEC-style restrictions) that support bounded memory and timing evidence.

Finally, FMI already provides a strong and widely accepted foundation for cross-tool interoperability in high-fidelity simulation, and its evolution toward FMI 3.0, together with embedded-oriented initiatives such as eFMI, outlines a clear trajectory toward embedded actuator deployment. Realizing this potential at industrial scale now hinges on execution-level evidence: reproducible benchmarking, conformance suites that stress timing and stiffness/event density, and qualification-oriented toolchains that can justify deterministic, resource-bounded, and compliant execution without compromising the interoperability benefits that motivated the standard in the first place. The opportunity is timely: the semantic hooks for time-aware execution now exist (FMI 3.0), and the path toward analyzable deployment is becoming concrete (eFMI/GALEC). The next decisive step is to standardize and publish end-to-end evidence that connects these semantics to measured real-time behavior in actuator-relevant benchmarks and industrial reference workflows.

## Appendix A Supplementary Material

Supplementary S1 is provided in two parts. S1-A contains the PRISMA-lite qualitative synthesis corpus ($n=36$; $n=28$ from the initial search plus 8 new entries from the Web of Science supplementary search) and provides, for each included study, the structured extraction variables described in Sect. 2.4: FMI version/mode, execution platform class, real-time assumptions, actuator-relevant model characteristics, reported metrics or validation setting, source type (peer-reviewed/preprint flag), and dual-version annotation where applicable. Studies for which timing or resource data were not reported are explicitly flagged to distinguish demonstrated from inferred feasibility. S1-B contains the extended evidence inventory (non-PRISMA), covering normative sources, standards, implementers' guides, tooling documentation, and domain-adjacent contextual references that are excluded from PRISMA counts. A single *Link* field is provided: DOI is preferred when available; otherwise, the best-available URL is shown.

**Table A1** Supplementary S1-A: PRISMA-lite qualitative synthesis corpus ($n=36$; $n=28$ from initial search revised from $n=51$, plus 8 new entries from Web of Science supplementary search). Abbreviations—FMI mode: ME = Model Exchange; CS = Co-Simulation; SE = Scheduled Execution; eFMI = Embedded FMI code generation; DCP = Distributed Co-simulation Protocol. Platform: ECU = Electronic Control Unit; HIL = Hardware-in-the-Loop; MCU = Microcontroller Unit; RTOS = Real-Time Operating System; HPC = High-Performance Computing. [P] = preprint retained for unique implementation details unavailable elsewhere; excluded from quality-scored synthesis

| ID | Yr | Title | FMI Ver./Mode | Platform | RT / Sched. Assumptions | Actuator / Model Focus | Evidence Type | Flags |
|---|---|---|---|---|---|---|---|---|
| S1-028 | 2015 | Test of Basic Co-Simulation Algorithms Using FMI https://doi.org/10.3384/ecp15118865 | FMI 2.0 / CS | Desktop | Fixed-step; communication-step stability analysis; Jacobi vs. Gauss–Seidel | Generic numerical co-simulation; master algorithm behaviour | Benchmark / simulation | Peer-reviewed (conf.) |
| S1-029 | 2017 | FMI-Based Validation of Embedded Control Units in Industrial Applications https://doi.org/10.3384/ecp17132307 | FMI 2.0 / CS | ECU / HIL | HIL timing; industrial SIL/HIL validation pipeline | Embedded control units; plant-model reference in SIL/HIL | HIL validation | Peer-reviewed (conf.) |
| S1-030 | 2018 | Co-simulation: A Survey https://doi.org/10.1145/3179993 | FMI 2.0 / ME, CS | Desktop | Numerical stability; master algorithm analysis; coupling semantics | Generic dynamic systems; co-simulation coupling and determinism | Survey / conceptual | Peer-reviewed (journal) |
| S1-031 | 2019 | Evaluation of Co-Simulation Master Algorithms in MasterSim https://doi.org/10.3384/ecp19157769 | FMI 2.0 / CS | Desktop | Step-size control; stability; rollback support assessment | Generic coupled systems; orchestration policy evaluation | Benchmark | Peer-reviewed (conf.) |
| S1-032 | 2021 | Energy Optimization of Rail Vehicles Using FMI-Based Virtual Drives https://doi.org/10.3384/ecp21181681 | FMI 2.0 / CS (workflow-dep.) | Desktop / HIL | Drive-level simulation; lifecycle energy analysis | Electromechanical rail drives; energy and drive-level fidelity | Simulation / conceptual | Peer-reviewed (conf.) |
| S1-033 | 2021 | The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations (Modelica Conf.) https://doi.org/10.3384/ecp21118127 | FMI 3.0 / SE | Desktop / RTOS (concept) | Clock semantics; SE partitions; RTOS mapping concepts | Generic clocked/SE models; timing-aware FMI integration | Conceptual / specification | Peer-reviewed (conf.); FMI 3.0 conceptual; no empirical clock/SE benchmark |
| S1-034 | 2021 | eFMI: Embedded FMI for Deterministic Code Generation https://doi.org/10.3384/ecp2118157 | eFMI / GALEC | ECU / Embedded | Deterministic code generation; bounded memory; WCET-oriented GALEC IR | Physical equations to analyzable embedded C via GALEC | Industrial prototype / pilot | Peer-reviewed (conf.) |
| S1-036 | 2021 | Advanced Controller Development Based on eFMI: Automotive Vertical Dynamics https://doi.org/10.3390/act10110301 | eFMI / code-gen | ECU / AUTOSAR | Vertical dynamics control; WCET-oriented; AUTOSAR integration | Automotive suspension actuator; active ride-height control | Industrial prototype | Peer-reviewed (journal) |
| S1-037 | 2022 | FMI-Based Co-Simulation of Actuator Dynamics in Robotic Systems Using Gazebo https://doi.org/10.20906/CBA2022/3210 | FMI 2.0 / CS | Desktop / ROS–Gazebo | Simulation loop timing; closed-loop co-simulation with physics engine | Joint actuators; friction, compliance, and robot dynamics | Simulation / prototype | Peer-reviewed (conf.) |
| S1-038 | 2022 | The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations (Electronics) https://doi.org/10.3390/electronics11213635 | FMI 3.0 / ME, CS, SE | Desktop | Clock semantics; SE concepts; discrete-time control loops | Generic / control systems; timing-aware FMI interface | Conceptual / survey | Peer-reviewed (journal); FMI 3.0 conceptual; no empirical clock/SE benchmark |

**Table A1** (continued)

| ID | Yr | Title | FMI Ver./Mode | Platform | RT / Sched. Assumptions | Actuator / Model Focus | Evidence Type | Flags |
|---|---|---|---|---|---|---|---|---|
| S1-039 | 2023 | Secure Exchange of Black-Box Simulation Models using FMI https://doi.org/10.3384/ecp204487 | FMI 2.0–3.0 / CS | Desktop / industrial | IP protection; secure model exchange; not RT-focused | Black-box industrial FMUs; supply-chain security and IP management | Conceptual / prototype | Peer-reviewed (conf.); duplicate S1-312 removed (same DOI) |
| S1-040 | 2025 | Classification of Edge and Embedded Platforms for Simulation/Control Workloads (ResearchGate—URL in S1-B) | Platform agnostic | Edge / MCU (survey) | Compute / memory benchmarking; embedded target characterisation | Various workloads on constrained hardware; platform taxonomy | Benchmark (embedded) | **[P]** Preprint (ResearchGate); no peer-reviewed version found |
| S1-041 | 2025 | Towards Native FMI Support in ROS–Gazebo Pipelines https://doi.org/10.3384/ecp218 | FMI 2.0–3.0 / CS | Desktop / ROS–Gazebo | ROS control loop timing; FMI-middleware integration latency | Robot actuator dynamics; joint-level FMU integration in ROS 2 | Prototype | Peer-reviewed (conf.) |
| S1-042 | 2025 | CDL-PLC Translator: Modelica HVAC Control to IEC 61,131–3 PLC https://doi.org/10.3384/ecp218515 | FMI 2.0 / Modelica to PLC | PLC (IEC 61,131–3) | PLC execution semantics; model-to-PLC deployment gap | Actuator/drive control logic; embedded PLC execution | Prototype / translation | Peer-reviewed (conf.) |
| S1-044 | 2025 | Executing FMI 2.0 FMUs on Embedded Platforms Using MicroPython https://doi.org/10.3384/ecp218535 | FMI 2.0 / CS (step-wise) | MCU (ESP32) | Step-time on MCU; memory footprint; non-stiff model feasibility | Non-stiff dynamic models executed on lightweight MicroPython runtime | Embedded prototype / benchmark | Peer-reviewed (conf.) |
| S1-045 | 2025 | FMI-Based Distributed Co-Simulation with Security and IP Safeguards https://doi.org/10.48550/arXiv.2510.20403 | FMI 2.0–3.0 / CS | Desktop / distributed | Distributed co-simulation; secure multi-party execution | Multi-FMU systems; IP-protected exchange across organisational boundaries | Prototype | **[P]** Preprint (arXiv); no peer-reviewed version at time of search |
| S1-266 | 2023 | Object-Oriented Modelling of Flexible Cables (Absolute Node Coordinates) https://doi.org/10.3384/ecp20453 | FMI 2.0 / ME (Modelica) | Desktop | Not addressed; offline simulation | Flexible cables / actuator compliance and structural compliance modelling | Simulation | Peer-reviewed (conf.) |
| S1-269 | 2023 | Distributed Parameter Pneumatics https://doi.org/10.3384/ecp20485 | FMI 2.0 | Desktop | Not addressed | Pneumatic actuator dynamics; distributed-parameter model fidelity | Simulation | Peer-reviewed (conf.) |
| S1-274 | 2023 | Paving the Way for Hybrid Twins Using Neural Functional Mock-up Units https://doi.org/10.3384/ecp204141 | FMI 2.0–3.0 | Desktop / edge (concept) | Not addressed (conceptual) | FMI + ML hybrid; data-driven actuator model augmentation via neural FMUs | Conceptual | Peer-reviewed (conf.) |
| S1-277 | 2023 | Introducing Dialectic Mechanics https://doi.org/10.3384/ecp204167 | Modelica / ME | Desktop | Foundational mechanics; not RT-focused | Mechanical actuator foundations; constraint and contact handling | Simulation | Peer-reviewed (conf.) |

**Table A1** (continued)

| ID | Yr | Title | FMI Ver./Mode | Platform | RT / Sched. Assumptions | Actuator / Model Focus | Evidence Type | Flags |
|---|---|---|---|---|---|---|---|---|
| S1-284 | 2023 | Dialectic Mechanics: Extension for Real-Time Simulation https://doi.org/10.3384/ecp204239 | FMI 2.0 / CS (RT-ext.) | Desktop / RT target | Real-time simulation extension; fixed-step RT execution semantics | Mechanical systems; real-time constraint handling under fixed-step policy | Prototype / RT | Peer-reviewed (conf.) |
| S1-285 | 2023 | Testing V&V of DCP Interface for Distributed Real-Time Applications https://doi.org/10.3384/ecp204245 | DCP (FMI-related) | Desktop / distrib. RT | Distributed real-time validation; DCP standard semantics | Distributed control systems; V&V for RT execution | Prototype | Peer-reviewed (conf.) |
| S1-293 | 2023 | Import and Export of Functional Mock-up Units in CasADi https://doi.org/10.3384/ecp204321 | FMI 2.0 / ME | Desktop | Optimal control integration; not embedded-focused | FMU import/export for optimal control; actuator trajectory optimisation | Prototype | Peer-reviewed (conf.) |
| S1-300 | 2023 | Beyond FMI—Towards New Applications with Layered Standards https://doi.org/10.3384/ecp204381 | FMI 2.0–3.0 | Desktop / industrial | Ecosystem standardisation; layered FMI extensions | Generic FMI ecosystem and interoperability scope for industrial workflows | Conceptual | Peer-reviewed (conf.) |
| S1-311 | 2023 | A preCICE-FMI Runner to Couple FMUs to PDE-Based Simulations https://doi.org/10.3384/ecp204479 | FMI 2.0–3.0 / CS | Desktop / HPC | Multi-physics co-simulation; coupling latency and synchronisation | PDE + FMU coupling; structural/fluid–actuator interaction | Prototype / benchmark | Peer-reviewed (conf.) |
| S1-331 | 2023 | Race Car Cooling System Model for Real-Time Use in Driving Simulator https://doi.org/10.3384/ecp204675 | FMI 2.0 / CS | Desktop / RT simulator | Real-time driving simulator; fixed-step RT execution; step-time budget | Thermal management under RT constraints; embedded-adjacent RT scenario | RT simulation | Peer-reviewed (conf.) |
| S1-336 | 2023 | FMI- and SSP-Based Model Integration for Digital Twin of Railway Infrastructure https://doi.org/10.3384/ecp204717 | FMI 2.0–3.0 / CS | Desktop / digital twin | Railway infrastructure digital twin; lifecycle integration; system-level | Railway system DT; multi-domain FMI/SSP; actuator-inclusive lifecycle | Digital twin prototype | Peer-reviewed (conf.) |
| S1-339 | 2023 | Hybrid Power Systems Simulation and Optimization Using SSP and FMI https://doi.org/10.3384/ecp204743 | FMI 2.0–3.0 | Desktop | Not addressed | Power systems + FMI/SSP; multi-domain simulation and optimisation | Simulation | Peer-reviewed (conf.) |
| S1-350 | 2024 | MoRC: A Modular Robot Controller Based on FMI https://doi.org/10.1109/ICRA57147.2024.10611553 | FMI 2.0 / CS | Desktop / ROS 2 | ROS 2 control loop; modular FMU composition; real-time robot actuation | Robot joint actuators; modular FMU-based actuator and controller architecture | Prototype / system | Peer-reviewed (conf.); WoS supplementary |
| S1-351 | 2022 | Real-Time Emission Prediction with Detailed Chemistry under Real-World Boundary Conditions https://doi.org/10.3390/en15010261 | FMI 2.0 / CS | ECU / HIL | HIL real-time loop; engine emission model under RT constraints; fixed-step execution | Engine actuator dynamics;emission-relevant combustion and actuator behaviour in HIL | HIL validation | Peer-reviewed (journal); WoS supplementary |

**Table A1** (continued)

| ID | Yr | Title | FMI Ver./Mode | Platform | RT / Sched. Assumptions | Actuator / Model Focus | Evidence Type | Flags |
|---|---|---|---|---|---|---|---|---|
| S1-352 | 2025 | Multiphysics Modeling and Co-Simulation of Ship Electric Propulsion Drives https://doi.org/10.1109/TTE.2024.3480009 | FMI 2.0–3.0 / CS | Desktop / maritime | Ship propulsion drive dynamics; multi-domain electromechanical coupling | Electric propulsion drives; multi-physics actuator dynamics for maritime systems | Simulation / prototype | Peer-reviewed (journal); WoS supplementary; mixed FMI 2.0–3.0 label; no isolated clock/SE test reported |
| S1-353 | 2023 | Modeling and Simulating Wind Energy Generation Systems by Means of FMI Co-Simulation https://doi.org/10.3390/en16197013 | FMI 2.0 / CS | Desktop | Variable-step; wind turbine dynamics coupling; energy conversion modelling | Wind turbine drive-train and actuator dynamics; electromechanical conversion co-simulation | Simulation | Peer-reviewed (journal); WoS supplementary |
| S1-354 | 2026 | Functional Mock-up Interface Support in Project Chrono Multibody Solver https://doi.org/10.1007/s11044-025-10106-9 | FMI 2.0–3.0 / ME, CS | Desktop / HPC | Multibody solver integration; flexible body coupling; not RT-focused | Flexible mechanical bodies; actuator compliance and structural dynamics in multibody FMI co-simulation | Simulation / benchmark | Peer-reviewed (journal); WoS supplementary; mixed FMI 2.0–3.0 label; no embedded clock/SE timing evidence |
| S1-355 | 2025 | Adding Higher-Level Semantics to Functional Mock-up Units for Model Composition https://doi.org/10.1007/s10270-024-01244-3 | FMI 2.0–3.0 / ME, CS | Desktop | Compositional semantics; interface contracts; formal model composition | FMU composition and interface semantics; formal foundations for multi-domain co-simulation | Formal / conceptual | Peer-reviewed (journal); WoS supplementary; formal/conceptual; not empirical FMI 3.0 SE evidence |
| S1-356 | 2026 | FMUiL: An Open-Source Package for In-the-Loop Simulations with FMUs https://doi.org/10.1016/j.softx.2026.102560 | FMI 2.0–3.0 / CS | Desktop / embedded-adjacent | In-the-loop integration; step-wise FMU execution; real-time-adjacent testing | FMU integration for in-the-loop testing; toolchain for actuator-centric embedded simulation validation | Software / open-source | Peer-reviewed (journal); WoS supplementary; mixed FMI 2.0–3.0 tooling; no isolated clock/SE benchmark |
| S1-357 | 2025 | train_fmu_gym: A Functional Mock-up Unit-Based Framework for Reinforcement Learning of Multi-Physical Systems https://doi.org/10.1109/ACCESS.2025.3636787 | FMI 2.0–3.0 / CS | Desktop | RL training loop; FMU-based environment stepping; not real-time-critical | Multi-physical FMU environments for RL; actuator-relevant control policy training | Simulation / software | Peer-reviewed (journal); WoS supplementary; mixed FMI 2.0–3.0 software; not real-time-critical clock/SE evidence |

**Notes on S1-A revision.** 23 entries from the initial corpus ($n=51$) were reclassified to S1-B: 22 addressed non-actuator-centric domains (district heating, building systems, generic Modelica tooling, regression testing infrastructure—exclusion code E2), and 1 was a duplicate (S1-312, same DOI as S1-039—exclusion code E6). The revised corpus ($n=36$) requires explicit treatment of actuator or drive dynamics, embedded/real-time constraints, or direct application to industrial mechatronic or robotic actuation systems. Entries S1-350 to S1-357 were added from the Web of Science supplementary search (25 records retrieved; 1 duplicate of S1-038 removed; 12 excluded for non-actuator-centric domain E2; 4 reclassified to S1-B; 8 meeting inclusion criteria). **[P]** entries are preprints retained for unique implementation details unavailable elsewhere; they are excluded from quality-scored synthesis. For mixed FMI 2.0–3.0 entries, the label indicates declared compatibility or scope only; unless clocks or Scheduled Execution are explicitly evaluated, the entry is not counted as empirical FMI 3.0 timing-semantics evidence.

**Table A2** Supplementary S1-B: extended evidence inventory (non-PRISMA). DOI is shown when available; otherwise the best-available URL is provided

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-001 | 1997 | Computer-Controlled Systems: Theory and Design | https://books.google.com/books/about/Computer_Controlled_Systems.html?id=9Y6D5vviqMgC |
| S1-002 | 2001 | Control System Design | https://books.google.com/books/about/Control_System_Design.html?id=7dNSAAAAMAAJ |
| S1-003 | 2016 | Robotics: Modelling, Planning and Control | https://doi.org/10.1007/978-1-84628-642-1 |
| S1-004 | 2025 | 250 Tools supporting FMI listed on the FMI tools page! (News) | https://fmi-standard.org/news/2025-07-14-fmi-supported-by-250-tools/ |
| S1-005 | 2025 | FMI Tools and Projects | https://github.com/fmi-tools |
| S1-006 | 2024 | Functional Mock-up Interface (FMI) Specification 3.0.2 | https://fmi-standard.org/docs/3.0.2/ |
| S1-007 | 2025 | FMI 3.0 Implementers' Guide (living document) | https://modelica.github.io/fmi-guides/main/fmi-guide/ |
| S1-008 | 2025 | FMI Layered Standard for Network Communication (FMI-LS-BUS) v1.0.0 | https://fmi-standard.org/fmi-ls-bus/1.0.0/ |
| S1-009 | 2025 | Release of FMI Layered Standard for Network Communication (FMI-LS-BUS) v1.0 | https://fmi-standard.org/news/2025-07-21-fmi-ls-bus-v1.0-release/ |
| S1-010 | 2014 | Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0 | https://fmi-standard.org/docs/2.0/ |
| S1-011 | 2018 | Towards the Verification of Hybrid Co-simulation Algorithms | https://clagms.github.io/assets/pdfs/Thule2018.pdf |
| S1-012 | 2021 | eFMI (FMI for Embedded Systems) in AUTOSAR for Next Generation Automotive Software Development | https://doi.org/10.4271/2021-26-0048 |
| S1-013 | 2021 | Integrating eFMI-Based Models into AUTOSAR-Compliant Embedded Systems | https://doi.org/10.3390/act10110301 |
| S1-014 | 2022 | Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 3.0 | https://fmi-standard.org/docs/3.0/ |
| S1-015 | 2024 | eFMI Annual Project Report 2024 | https://www.efmi-standard.org/media/resources/MAP-eFMI-annual-project-report-2024.pdf |
| S1-016 | 2026 | Simcenter Amesim − 1D System Simulation | https://plm.sw.siemens.com/en-US/simcenter/systems-simulation/amesim/ |
| S1-017 | 2026 | FMI Blockset for Simulink | https://www.claytex.com/tools/fmi-blockset/ |
| S1-018 | 2026 | Dymola – Multi-Engineering Modeling and Simulation | https://www.3ds.com/products/catia/dymola |
| S1-019 | 2026 | Modelon Impact | https://modelon.com/modelon-impact/ |
| S1-020 | 2026 | MapleSim – Physical Modeling and Simulation | https://www.maplesoft.com/products/maplesim/ |
| S1-021 | 2026 | Orthogonal ODE – Modelica+AI MBSE Platform | https://www.orthogonal.dev |
| S1-022 | 2026 | OpenModelica – Open Source Modeling and Simulation Environment | https://openmodelica.org/ |
| S1-023 | 2026 | SimulationX – Physical System Simulation | https://www.simulationx.com |
| S1-024 | 2026 | ANSYS Twin Builder | https://www.ansys.com/products/digital-twin/ansys-twin-builder |
| S1-025 | 1995 | Series Elastic Actuators | https://ieeexplore.ieee.org/document/525827 |
| S1-026 | 2012 | Functional Mockup Interface 2.0: The Standard for Tool Independent Exchange of Simulation Models | https://doi.org/10.3384/ecp12076173 |
| S1-027 | 2013 | Determinate Composition of FMUs for Co-Simulation | https://people.kth.se/dbro/papers/broman-et-al-2013-fmu-co-simulation.pdf |
| S1-035 | 2021 | The PRISMA 2020 statement: an updated guideline for reporting systematic reviews | https://www.bmj.com/content/372/bmj.n71 |
| S1-043 | 2025 | Prototypical Control for the Digital Twin of Aircraft Environmental Control System | https://doi.org/10.3384/ecp21881 |
| S1-046 | 2024 | EMPHYSIS Public Demonstrator Summary / Public Report on eFMI-related outcomes (Deliverable D7.9) | https://emphysis.github.io/pages/downloads/emphysis-public-demonstrator-summary.pdf |
| S1-047 | 2026 | FMI Library – Open Source C Library for FMI Integration | https://github.com/modelon-community/fmi-library |
| S1-048 | 2026 | FMITerminalBlock – FMI to IEC 61499 Bridge | https://github.com/AIT-IES/fmi-terminal-block |
| S1-049 | 2026 | gazebo-fmi Plugin for Gazebo | https://github.com/robotology/gazebo-fmi |
| S1-050 | 2026 | fmi_adapter – FMI Adapter for ROS 2 | https://github.com/boschresearch/fmi_adapter |
| S1-051 | 2026 | ModelicaStandardLibrary | https://build.openmodelica.org/Documentation/ModelicaStandardLibrary.html |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-052 | 2026 | VehicleInterfaces | https://build.openmodelica.org/Documentation/VehicleInterfaces.html |
| S1-053 | 2026 | Modelica-Compliance | https://build.openmodelica.org/Documentation/Modelica-Compliance.html |
| S1-054 | 2026 | Modelica_LinearSystems2 | https://build.openmodelica.org/Documentation/Modelica_LinearSystems2.html |
| S1-055 | 2026 | ADGenKinetics | https://build.openmodelica.org/Documentation/ADGenKinetics.html |
| S1-056 | 2026 | ADMSL | https://build.openmodelica.org/Documentation/ADMSL.html |
| S1-057 | 2026 | AdvancedNoise | https://build.openmodelica.org/Documentation/AdvancedNoise.html |
| S1-058 | 2026 | AixLib | https://build.openmodelica.org/Documentation/AixLib.html |
| S1-059 | 2026 | AlgebraTestSuite | https://build.openmodelica.org/Documentation/AlgebraTestSuite.html |
| S1-060 | 2026 | ApproxSpline | https://build.openmodelica.org/Documentation/ApproxSpline.html |
| S1-061 | 2026 | ATplus | https://build.openmodelica.org/Documentation/ATplus.html |
| S1-062 | 2026 | BioChem | https://build.openmodelica.org/Documentation/BioChem.html |
| S1-063 | 2026 | BondGraph | https://build.openmodelica.org/Documentation/BondGraph.html |
| S1-064 | 2026 | BondLib | https://build.openmodelica.org/Documentation/BondLib.html |
| S1-065 | 2026 | BrineProp | https://build.openmodelica.org/Documentation/BrineProp.html |
| S1-066 | 2026 | BuildingControlLib | https://build.openmodelica.org/Documentation/BuildingControlLib.html |
| S1-067 | 2026 | Buildings | https://build.openmodelica.org/Documentation/Buildings.html |
| S1-068 | 2026 | BuildingSystems | https://build.openmodelica.org/Documentation/BuildingSystems.html |
| S1-069 | 2026 | BuildSysPro | https://build.openmodelica.org/Documentation/BuildSysPro.html |
| S1-070 | 2026 | BusinessSimulation | https://build.openmodelica.org/Documentation/BusinessSimulation.html |
| S1-071 | 2026 | Chemical | https://build.openmodelica.org/Documentation/Chemical.html |
| S1-072 | 2026 | ComplexLib | https://build.openmodelica.org/Documentation/ComplexLib.html |
| S1-073 | 2026 | DateTime | https://build.openmodelica.org/Documentation/DateTime.html |
| S1-074 | 2026 | DeployStructLib | https://build.openmodelica.org/Documentation/DeployStructLib.html |
| S1-075 | 2026 | DESLib | https://build.openmodelica.org/Documentation/DESLib.html |
| S1-076 | 2026 | dhcSim | https://build.openmodelica.org/Documentation/dhcSim.html |
| S1-077 | 2026 | DisHeatLib | https://build.openmodelica.org/Documentation/DisHeatLib.html |
| S1-078 | 2026 | DriveControl | https://build.openmodelica.org/Documentation/DriveControl.html |
| S1-079 | 2026 | EHPTlib | https://build.openmodelica.org/Documentation/EHPTlib.html |
| S1-080 | 2026 | ElectricalEnergyStorage | https://build.openmodelica.org/Documentation/ElectricalEnergyStorage.html |
| S1-081 | 2026 | electrolytemedia | https://build.openmodelica.org/Documentation/electrolytemedia.html |
| S1-082 | 2026 | ElectroMechanicalDrives | https://build.openmodelica.org/Documentation/ElectroMechanicalDrives.html |
| S1-083 | 2026 | EMBS_ModelicaLib | https://build.openmodelica.org/Documentation/EMBS_ModelicaLib.html |
| S1-084 | 2026 | EMOTH | https://build.openmodelica.org/Documentation/EMOTH.html |
| S1-085 | 2026 | ESL | https://build.openmodelica.org/Documentation/ESL.html |
| S1-086 | 2026 | ExtendedPetriNets | https://build.openmodelica.org/Documentation/ExtendedPetriNets.html |
| S1-087 | 2026 | ExternalMedia | https://build.openmodelica.org/Documentation/ExternalMedia.html |
| S1-088 | 2026 | ExternalMemoryLib | https://build.openmodelica.org/Documentation/ExternalMemoryLib.html |
| S1-089 | 2026 | ExternData | https://build.openmodelica.org/Documentation/ExternData.html |
| S1-090 | 2026 | FailureModes | https://build.openmodelica.org/Documentation/FailureModes.html |
| S1-091 | 2026 | FastBuildings | https://build.openmodelica.org/Documentation/FastBuildings.html |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-092 | 2026 | FaultTriggering | https://build.openmodelica.org/Documentation/FaultTriggering.html |
| S1-093 | 2026 | FCSys | https://build.openmodelica.org/Documentation/FCSys.html |
| S1-094 | 2026 | FeedDriveLibrary | https://build.openmodelica.org/Documentation/FeedDriveLibrary.html |
| S1-095 | 2026 | FluidSystemComponents | https://build.openmodelica.org/Documentation/FluidSystemComponents.html |
| S1-096 | 2026 | FMITest | https://build.openmodelica.org/Documentation/FMITest.html |
| S1-097 | 2026 | FreeFluidsModelica | https://build.openmodelica.org/Documentation/FreeFluidsModelica.html |
| S1-098 | 2026 | FuelCellLib | https://build.openmodelica.org/Documentation/FuelCellLib.html |
| S1-099 | 2026 | FuzzyControl | https://build.openmodelica.org/Documentation/FuzzyControl.html |
| S1-100 | 2026 | Greenhouses-Library | https://build.openmodelica.org/Documentation/Greenhouses-Library.html |
| S1-101 | 2026 | HanserModelica | https://build.openmodelica.org/Documentation/HanserModelica.html |
| S1-102 | 2026 | HeatTransferComponents | https://build.openmodelica.org/Documentation/HeatTransferComponents.html |
| S1-103 | 2026 | HelmholtzMedia | https://build.openmodelica.org/Documentation/HelmholtzMedia.html |
| S1-104 | 2026 | Icons | https://build.openmodelica.org/Documentation/Icons.html |
| S1-105 | 2026 | IdealizedContact | https://build.openmodelica.org/Documentation/IdealizedContact.html |
| S1-106 | 2026 | IDEAS | https://build.openmodelica.org/Documentation/IDEAS.html |
| S1-107 | 2026 | IndustrialControlSystems | https://build.openmodelica.org/Documentation/IndustrialControlSystems.html |
| S1-108 | 2026 | ipsl | https://build.openmodelica.org/Documentation/ipsl.html |
| S1-109 | 2026 | KeyWordIO | https://build.openmodelica.org/Documentation/KeyWordIO.html |
| S1-110 | 2026 | LCC_HVDC | https://build.openmodelica.org/Documentation/LCC_HVDC.html |
| S1-111 | 2026 | LibRAS | https://build.openmodelica.org/Documentation/LibRAS.html |
| S1-112 | 2026 | LinearMPC | https://build.openmodelica.org/Documentation/LinearMPC.html |
| S1-113 | 2026 | Modelica-Arduino | https://build.openmodelica.org/Documentation/Modelica-Arduino.html |
| S1-114 | 2026 | Modelica-Drone-3D-FMI | https://build.openmodelica.org/Documentation/Modelica-Drone-3D-FMI.html |
| S1-115 | 2026 | Modelica-ExternalLibrary | https://build.openmodelica.org/Documentation/Modelica-ExternalLibrary.html |
| S1-116 | 2026 | modelica-flight | https://build.openmodelica.org/Documentation/modelica-flight.html |
| S1-117 | 2026 | Modelica-GNU_ScientificLibrary | https://build.openmodelica.org/Documentation/Modelica-GNU_ScientificLibrary.html |
| S1-118 | 2026 | modelica-ibpsa | https://build.openmodelica.org/Documentation/modelica-ibpsa.html |
| S1-119 | 2026 | Modelica-MVEM | https://build.openmodelica.org/Documentation/Modelica-MVEM.html |
| S1-120 | 2026 | ModelicaADS | https://build.openmodelica.org/Documentation/ModelicaADS.html |
| S1-121 | 2026 | ModelicaDEVS | https://build.openmodelica.org/Documentation/ModelicaDEVS.html |
| S1-122 | 2026 | ModelicaDFR | https://build.openmodelica.org/Documentation/ModelicaDFR.html |
| S1-123 | 2026 | ModelicaTableAdditions | https://build.openmodelica.org/Documentation/ModelicaTableAdditions.html |
| S1-124 | 2026 | ModelicaTutorials | https://build.openmodelica.org/Documentation/ModelicaTutorials.html |
| S1-125 | 2026 | Modelica_DeviceDrivers | https://build.openmodelica.org/Documentation/Modelica_DeviceDrivers.html |
| S1-126 | 2026 | Modelica_Requirements | https://build.openmodelica.org/Documentation/Modelica_Requirements.html |
| S1-127 | 2026 | Modelica_StateGraph2 | https://build.openmodelica.org/Documentation/Modelica_StateGraph2.html |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-128 | 2026 | ModPowerSystems | https://build.openmodelica.org/Documentation/ModPowerSystems.html |
| S1-129 | 2026 | MoMoUrEnSySi | https://build.openmodelica.org/Documentation/MoMoUrEnSySi.html |
| S1-130 | 2026 | MoSDH | https://build.openmodelica.org/Documentation/MoSDH.html |
| S1-131 | 2026 | MotorcycleDynamics | https://build.openmodelica.org/Documentation/MotorcycleDynamics.html |
| S1-132 | 2026 | MotorcycleLib | https://build.openmodelica.org/Documentation/MotorcycleLib.html |
| S1-133 | 2026 | msgpack-modelica | https://build.openmodelica.org/Documentation/msgpack-modelica.html |
| S1-134 | 2026 | MultiBondLib | https://build.openmodelica.org/Documentation/MultiBondLib.html |
| S1-135 | 2026 | MultiPhaseMixtureMedia | https://build.openmodelica.org/Documentation/MultiPhaseMixtureMedia.html |
| S1-136 | 2026 | NCLib | https://build.openmodelica.org/Documentation/NCLib.html |
| S1-137 | 2026 | netCDF-DataReader | https://build.openmodelica.org/Documentation/netCDF-DataReader.html |
| S1-138 | 2026 | NeuralNetwork | https://build.openmodelica.org/Documentation/NeuralNetwork.html |
| S1-139 | 2026 | Noise | https://build.openmodelica.org/Documentation/Noise.html |
| S1-140 | 2026 | ObjectStab | https://build.openmodelica.org/Documentation/ObjectStab.html |
| S1-141 | 2026 | open-bldc-modelica | https://build.openmodelica.org/Documentation/open-bldc-modelica.html |
| S1-142 | 2026 | openfdm | https://build.openmodelica.org/Documentation/openfdm.html |
| S1-143 | 2026 | OpenHydraulics | https://build.openmodelica.org/Documentation/OpenHydraulics.html |
| S1-144 | 2026 | OpenIMDML | https://build.openmodelica.org/Documentation/OpenIMDML.html |
| S1-145 | 2026 | OpenIPSL | https://build.openmodelica.org/Documentation/OpenIPSL.html |
| S1-146 | 2026 | OpenModelica_Microgrids | https://build.openmodelica.org/Documentation/OpenModelica_Microgrids.html |
| S1-147 | 2026 | Optimisers | https://build.openmodelica.org/Documentation/Optimisers.html |
| S1-148 | 2026 | PDELib | https://build.openmodelica.org/Documentation/PDELib.html |
| S1-149 | 2026 | Pharmacolibrary | https://build.openmodelica.org/Documentation/Pharmacolibrary.html |
| S1-150 | 2026 | PhotoVoltaics | https://build.openmodelica.org/Documentation/PhotoVoltaics.html |
| S1-151 | 2026 | Physiolibrary | https://build.openmodelica.org/Documentation/Physiolibrary.html |
| S1-152 | 2026 | Physiomodel | https://build.openmodelica.org/Documentation/Physiomodel.html |
| S1-153 | 2026 | PlanarMechanics | https://build.openmodelica.org/Documentation/PlanarMechanics.html |
| S1-154 | 2026 | PNlib | https://build.openmodelica.org/Documentation/PNlib.html |
| S1-155 | 2026 | InstantaneousSymmetricalComponents | https://build.openmodelica.org/Documentation/InstantaneousSymmetricalComponents.html |
| S1-156 | 2026 | ModelicaAdditions | https://build.openmodelica.org/Documentation/ModelicaAdditions.html |
| S1-157 | 2026 | Modelica_ElectricalEnergyStorages | https://build.openmodelica.org/Documentation/Modelica_ElectricalEnergyStorages.html |
| S1-158 | 2026 | Modelica_Electrical_PowerConverters | https://build.openmodelica.org/Documentation/Modelica_Electrical_PowerConverters.html |
| S1-159 | 2026 | Modelica_Fluid | https://build.openmodelica.org/Documentation/Modelica_Fluid.html |
| S1-160 | 2026 | Modelica_FundamentalWave | https://build.openmodelica.org/Documentation/Modelica_FundamentalWave.html |
| S1-161 | 2026 | Modelica_Magnetic | https://build.openmodelica.org/Documentation/Modelica_Magnetic.html |
| S1-162 | 2026 | Modelica_PowerFlow | https://build.openmodelica.org/Documentation/Modelica_PowerFlow.html |
| S1-163 | 2026 | Modelica_QuasiStatic_FluxTubes | https://build.openmodelica.org/Documentation/Modelica_QuasiStatic_FluxTubes.html |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-164 | 2026 | Modelica_Quasistationary | https://build.openmodelica.org/Documentation/Modelica_Quasistationary.html |
| S1-165 | 2026 | Modelica_Synchronous | https://build.openmodelica.org/Documentation/Modelica_Synchronous.html |
| S1-166 | 2026 | MultiBody | https://build.openmodelica.org/Documentation/MultiBody.html |
| S1-167 | 2026 | ObsoleteModelica3 | https://build.openmodelica.org/Documentation/ObsoleteModelica3.html |
| S1-168 | 2026 | QuasiStaticFundamentalWave | https://build.openmodelica.org/Documentation/QuasiStaticFundamentalWave.html |
| S1-169 | 2026 | Air Conditioning Library | https://www.modelon.com |
| S1-170 | 2026 | Aircraft Dynamics Library | https://www.modelon.com |
| S1-171 | 2026 | Alternative Vehicles Library | https://www.dlr.de |
| S1-172 | 2026 | Battery Library | https://www.3ds.com |
| S1-173 | 2026 | Belts Library | https://www.ltx.de |
| S1-174 | 2026 | BG_RT Library | https://www.ltx.de |
| S1-175 | 2026 | BLDCD – Brushless DC Drives | https://www.3ds.com |
| S1-176 | 2026 | Blue Energy Library | https://www.blue-energy-library.com |
| S1-177 | 2026 | ClaRa+ Library | https://www.claralib.com |
| S1-178 | 2026 | Cooling Library | https://www.3ds.com |
| S1-179 | 2026 | EDrives Library | https://edrives.eu |
| S1-180 | 2026 | Electric Power Library | https://www.modelon.com |
| S1-181 | 2026 | Electrification Library | https://www.modelon.com |
| S1-182 | 2026 | Energy Systems Library | https://www.modelon.com |
| S1-183 | 2026 | Engine Dynamics Library | https://www.modelon.com |
| S1-184 | 2026 | Environmental Control Library | https://www.modelon.com |
| S1-185 | 2026 | EPTL - Electrified Powertrains Library | https://www.3ds.com |
| S1-186 | 2026 | Flexible Bodies Library | https://vsdc.de |
| S1-187 | 2026 | FluidDynamics Library | https://www.xrg-simulation.de |
| S1-188 | 2026 | Fluid Power Library | https://www.claytex.com |
| S1-189 | 2026 | Fuel Cell Library | https://www.modelon.com |
| S1-190 | 2026 | Fuel Systems Library | https://www.modelon.com |
| S1-191 | 2026 | Green City Library | https://www.green-city-library.de |
| S1-192 | 2026 | Heat Exchanger Library | https://www.modelon.com |
| S1-193 | 2026 | HumanComfort Library | https://www.xrg-simulation.de |
| S1-194 | 2026 | HVAC Library | https://www.xrg-simulation.de |
| S1-195 | 2026 | Hydraulics Library | https://www.modelon.com |
| S1-196 | 2026 | Hydro Power Library | https://www.modelon.com |
| S1-197 | 2026 | Hydrogen Library | https://www.3ds.com |
| S1-198 | 2026 | Jet Propulsion Library | https://www.modelon.com |
| S1-199 | 2026 | Liquid Cooling Library | https://www.modelon.com |
| S1-200 | 2026 | OPCClassic | https://www.wolfram.com |
| S1-201 | 2026 | Optimization library | https://vsdc.de |
| S1-202 | 2026 | Pneumatics Library | https://www.modelon.com |
| S1-203 | 2026 | Power Train Library | https://vsdc.de |
| S1-204 | 2026 | Process Systems Library | https://tlk-energy.de |
| S1-205 | 2026 | ProcessingLine Library | https://en.dofware.com |
| S1-206 | 2026 | Rexroth BRSL | https://www.boschrexroth.com |
| S1-207 | 2026 | SmartCooling | https://www.wolfram.com |
| S1-208 | 2026 | SMArtInt | https://build.openmodelica.org/Documentation/SMArtInt.html |
| S1-209 | 2026 | Smart Electric Drives Library | https://www.3ds.com |
| S1-210 | 2026 | Statistics Library | https://www.eas.iis.fraunhofer.de |
| S1-211 | 2026 | Thermal Power Library | https://www.modelon.com |
| S1-212 | 2026 | TIL | https://www.tlk-thermo.com |
| S1-213 | 2026 | TILMedia | https://tilmedia.tlk-thermo.com |
| S1-214 | 2026 | UAV Dynamics Library | https://www.claytex.com |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-215 | 2026 | Vapor Cycle Library | https://www.modelon.com |
| S1-216 | 2026 | Vehicle Dynamics Library | https://www.modelon.com |
| S1-217 | 2026 | VeSyMA | https://www.claytex.com |
| S1-218 | 2026 | VeSyMA - Engines | https://www.claytex.com |
| S1-219 | 2026 | VeSyMA - Driver-in-the-Loop | https://www.claytex.com |
| S1-220 | 2026 | VeSyMA - Motorsports | https://www.claytex.com |
| S1-221 | 2026 | VeSyMA - Powertrain | https://www.claytex.com |
| S1-222 | 2026 | VeSyMA - Suspensions | https://www.claytex.com |
| S1-223 | 2026 | VeSyMA - Terrain Server | https://www.claytex.com |
| S1-224 | 2026 | Visualization Library | https://scil.dlr.de |
| S1-225 | 2026 | Wind Power Library (WPL) | https://www.3ds.com |
| S1-226 | 2026 | Wolfram Hydraulic | https://www.wolfram.com |
| S1-227 | 2026 | OpenModelica | https://openmodelica.org/ |
| S1-228 | 2026 | Altair Twin Activate | https://altair.com/twin-activate |
| S1-229 | 2026 | ANSYS Simplorer | https://www.ansys.com/products/systems/ansys-simplorer |
| S1-230 | 2026 | Dymola | https://www.dymola.com/ |
| S1-231 | 2026 | Systems Simulation Designer | https://www.youtube.com/watch?v=KBXfaUZkxfY |
| S1-232 | 2026 | SimulationX | https://www.simulationx.com/ |
| S1-233 | 2026 | MapleSim | https://www.maplesoft.com/products/maplesim/ |
| S1-234 | 2026 | ODE (Orthogonal) | https://www.orthogonal.dev/ |
| S1-235 | 2026 | Simcenter Amesim | https://plm.sw.siemens.com/en-US/simcenter/systems-simulation/amesim/ |
| S1-236 | 2026 | MWorks (Suzhou Tongyuan) | https://www.tongyuan.cc/ |
| S1-237 | 2026 | Wolfram SystemModeler | https://www.wolfram.com/system-modeler/ |
| S1-238 | 2026 | Atom (language-modelica package) | https://atom.io/ |
| S1-239 | 2026 | Modelica mode for Emacs | https://github.com/modelica-tools/modelica-mode |
| S1-240 | 2026 | Modelica Sublime Text Package | https://borischumichev.github.io/modelicaSublimeTextPackage/ |
| S1-241 | 2026 | UltraEdit Modelica syntax highlighting (WordfileModelica.txt) | https://modelica.org/tools/WordfileModelica.txt |
| S1-242 | 2026 | Visual Studio Code Modelica extension | https://marketplace.visualstudio.com/items?itemName=SimplyDanny.modelica |
| S1-243 | 2026 | Modelica Compliance Suite | https://github.com/modelica/Modelica-Compliance |
| S1-244 | 2026 | CSV Result Compare tool | https://github.com/modelica-tools/csv-compare |
| S1-245 | 2026 | MapleSim Standalone Modelica Parser (ModelicaSyntaxChecker) | https://github.com/modelica-tools/ModelicaSyntaxChecker |
| S1-246 | 2026 | MapleSim Modelica Checker (online) | http://www.maplesoft.com/products/maplesim/modelicachecker |
| S1-247 | 2026 | Trimming of trailing white spaces / Modelica cleanup | https://github.com/simulatino/trimtrailingwhitespaces |
| S1-248 | 2026 | PMD (Modelica support) | https://github.com/pmd/pmd |
| S1-249 | 2026 | awesim | https://github.com/saroele/awesim |
| S1-250 | 2026 | BuildingsPy | https://simulationresearch.lbl.gov/modelica/buildingspy |
| S1-251 | 2026 | DyMat | https://www.j-raedler.de/projects/dymat/ |
| S1-252 | 2026 | Highlight (syntax highlighting tool) | https://www.andre-simon.de/ |
| S1-253 | 2026 | listings-modelica | https://github.com/modelica-tools/listings-modelica |
| S1-254 | 2026 | ModelicaRes | https://kdavies4.github.io/ModelicaRes/ |
| S1-255 | 2026 | modelica-builder | https://github.com/urbanopt/modelica-builder |
| S1-256 | 2026 | modelica-fmt | https://github.com/urbanopt/modelica-fmt |
| S1-257 | 2026 | OpenModelica Microgrid Gym (OMG) | https://github.com/upb-lea/openmodelica-microgrid-gym |
| S1-258 | 2026 | PlotXY | https://ceraolo-plotxy.ing.unipi.it/ |
| S1-259 | 2026 | PySimulator | https://github.com/PySimulator/PySimulator |
| S1-260 | 2026 | SaveToFile.zip (Simulink-like To File block writing Dymola-compatible.mat) | https://modelica.org/tools/SaveToFile.zip |
| S1-261 | 2024 | eFMI Standard 1.0.0 Beta 1 | https://www.efmi-standard.org/media/home/eFMI-Standard-1.0.0-Beta-1.html |
| S1-287 | 2023 | Object-Oriented Formulation and Simulation of Models using Linear | https://doi.org/10.3384/ecp204265 |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-288 | 2023 | Accelerating the simulation of equation-based models by replacing non-linear algebraic loops with error-controlled machine learning surrogates | https://doi.org/10.3384/ecp204275 |
| S1-289 | 2023 | Exploiting Modelica and the OpenIPSL for University Campus | https://doi.org/10.3384/ecp204285 |
| S1-290 | 2023 | Application of the OpenModelica-MATLAB Interface to Integrated Simulation and Successive Linearization Based Model | https://doi.org/10.3384/ecp204293 |
| S1-291 | 2023 | Parameter Estimation of Modelica Building Models Using CasADi | https://doi.org/10.3384/ecp204301 |
| S1-292 | 2023 | Presentation, Validation and Application of the EnergyProcess Library | https://doi.org/10.3384/ecp204311 |
| S1-294 | 2023 | On the Characteristics of One-Dimensional Compressible Flow | https://doi.org/10.3384/ecp204327 |
| S1-295 | 2023 | LargeTESModelingToolkit: A Modelica Library for Large-scale Thermal Energy Storage Modeling and Simulation | https://doi.org/10.3384/ecp204337 |
| S1-296 | 2023 | Dynamic Modeling and Experimental Validation of Dishwasher with Heat | https://doi.org/10.3384/ecp204347 |
| S1-297 | 2023 | A Library to Simulate Processes in the Factory Hall | https://doi.org/10.3384/ecp204357 |
| S1-298 | 2023 | Modelica 3.6 - Changes, Benefits and Implementation | https://doi.org/10.3384/ecp204367 |
| S1-299 | 2023 | Modelica Models in SSP | https://doi.org/10.3384/ecp204375 |
| S1-301 | 2023 | Low-order aquifer thermal energy storage model for geothermal system simulation | https://doi.org/10.3384/ecp204389 |
| S1-302 | 2023 | Heat Consumer Model for Robust and Fast Simulations of District | https://doi.org/10.3384/ecp204397 |
| S1-303 | 2023 | Convection of Chemicals and Other Substances with ThermoSysPro | https://doi.org/10.3384/ecp204407 |
| S1-304 | 2023 | Using the DLR Thermofluid Stream Library for Thermal Management of Fuel Cell Systems in Aviation | https://doi.org/10.3384/ecp204415 |
| S1-305 | 2023 | Supporting Infinitely Fast Processes in Continuous System Modeling | https://doi.org/10.3384/ecp204423 |
| S1-306 | 2023 | A Modelica Library to Add Contact Dynamics and Terramechanics to Multi-Body Mechanics | https://doi.org/10.3384/ecp204433 |
| S1-308 | 2023 | Design ideas behind Bioprocess Library for Modelica | https://doi.org/10.3384/ecp204453 |
| S1-309 | 2023 | Creating cardiovascular and respiratory models using Physiolibrary 3.0 | https://doi.org/10.3384/ecp204463 |
| S1-310 | 2023 | Design proposal of a standardized Base Modelica language | https://doi.org/10.3384/ecp204469 |
| S1-313 | 2023 | The Common Requirement Modeling Language | https://doi.org/10.3384/ecp204497 |
| S1-314 | 2023 | Variable Structure System Simulation via Predefined Acausal Components | https://doi.org/10.3384/ecp204511 |
| S1-315 | 2023 | MoCITempGen: Modelica Continuous Integration Template Generator | https://doi.org/10.3384/ecp204521 |
| S1-316 | 2023 | Steady State and Dynamic Simulation of a Small-Scale Hollow | https://doi.org/10.3384/ecp204531 |
| S1-317 | 2023 | Modeling Components of a Turbine-Generator System for Sub-Synchronous Oscillation Studies with Modelica | https://doi.org/10.3384/ecp204541 |
| S1-318 | 2023 | DroneLibrary: Multi-domain Drone Modeling in Modelica | https://doi.org/10.3384/ecp204551 |
| S1-320 | 2023 | Simulation Study of Flow Instability in Parallel Multi-Channel Systems Based on Modelica | https://doi.org/10.3384/ecp204569 |
| S1-321 | 2023 | Integration of Heat Flow through Borders between Adjacent Zones in AixLib's Reduced-Order Model | https://doi.org/10.3384/ecp204577 |
| S1-322 | 2023 | Electrode boiler model for ancillary service simulation | https://doi.org/10.3384/ecp204589 |
| S1-323 | 2023 | Heat Exchanger Surrogates for a Vapor Compression System | https://doi.org/10.3384/ecp204599 |

**Table A2** (continued)

| ID | Yr | Title | Link |
|---|---|---|---|
| S1-324 | 2023 | 5th Generation District Heating and Cooling Modelica Models for | https://doi.org/10.3384/ecp204607 |
| S1-325 | 2023 | Status of the ClaRa Library: Detailed Transient Simulation of | https://doi.org/10.3384/ecp204617 |
| S1-326 | 2023 | Open-Source Models for Sand-Based Thermal Energy Storage in | https://doi.org/10.3384/ecp204627 |
| S1-328 | 2023 | Dropwise Condensation Water Drainage Model | https://doi.org/10.3384/ecp204643 |
| S1-329 | 2023 | Modeling Specialized Electric Power Generators, Excitation Systems and Prime Movers used by North American Utilities | https://doi.org/10.3384/ecp204653 |
| S1-330 | 2023 | Numerically Efficient Degradation Model of Catalyst Layers in PEM Fuel Cells using Modelica | https://doi.org/10.3384/ecp204665 |
| S1-332 | 2023 | Switching and Averaging Models of a Bidirectional, Half-Bridge Based DC-DC converter with Load Distribution | https://doi.org/10.3384/ecp204683 |
| S1-333 | 2023 | Mass Conservation in Vapor Compression Cycles: A Method for Ensuring Consistency with Redundant Dynamic States | https://doi.org/10.3384/ecp204693 |
| S1-334 | 2023 | Calibration Workflow for Mechanical and Thermal Applications | https://doi.org/10.3384/ecp204703 |
| S1-335 | 2023 | SSP in a Modelica Environment | https://doi.org/10.3384/ecp204711 |
| S1-337 | 2023 | PNRG – A Library for Modeling Variable Structure Energy Grids in Modelica using Energetic Petri Nets Christian Gutsche1, 2 Zizhe Wang1, 2 | https://doi.org/10.3384/ecp204727 |
| S1-338 | 2023 | Energy Efficiency Measures for Existing Factory Buildings | https://doi.org/10.3384/ecp204737 |
| S1-340 | 2023 | Information Classification: Internal INTERNAL & PARTNERS 5acXjzUk Simulation of Vehicle Headlamp Levelling systems | https://doi.org/10.3384/ecp204751 |
| S1-341 | 2023 | Piecewise-Steady-State Modelica Simulations for the Conceptual Design Phase of Industrial Processes | https://doi.org/10.3384/ecp204757 |
| S1-342 | 2023 | Modeling and Control Design of an | https://doi.org/10.3384/ecp204763 |
| S1-343 | 2023 | Coupling of Thermal and Electrical Systems for the Simulation of ECS Architectures | https://doi.org/10.3384/ecp204771 |
| S1-344 | 2023 | A Penalty Function-based Modelica Library for Multi-body | https://doi.org/10.3384/ecp204779 |
| S1-345 | 2023 | Automatic Optimization of Energy Supply Systems in Buildings and City Quarters based on Modelica Models | https://doi.org/10.3384/ecp204789 |
| S1-362 | 2023 | Hybrid Fire Testing Using FMI-Based Co-Simulation | https://doi.org/10.1016/j.firesaf.2023.103832 |
| S1-363 | 2024 | A Co-Simulation Approach to Onboard Support of Marine Operations | https://doi.org/10.1177/00375497241228623 |
| S1-364 | 2025 | Simulation Model Exchange in Process Industry: Requirements and Challenges | https://doi.org/10.1002/ceat.202400331 |
| S1-365 | 2024 | Integration of Geometry Modelling and Behaviour Simulation Based on Digital Layers | https://doi.org/10.1016/j.procir.2024.06.025 |

**Author contributions** S.R.O. conceived the study, performed the systematic review and taxonomy coding, and drafted the manuscript. M.S., I.Y., and E.Z. contributed to the technical framing, critical revisions, and supervision. All authors reviewed and approved the final manuscript.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was supported by the SENDOA project ID: KK-2025/00102; the DBaskIN KK-2025/00012 project, financed by the Council of the Basque Country.

**Data availability** No datasets were generated or analysed during the current study.

## Declarations

**Ethics Approval** Not applicable. This study is a systematic literature review and does not involve human participants, animals, or personal data.

**Competing Interests** The authors declare no competing interests.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References

1. Siciliano B, Sciavicco L, Villani L, Oriolo G (2016) Robotics: modelling, planning and control. Springer. https://doi.org/10.1007/978-1-84628-642-1
2. Pratt GA, Williamson MM (1995) Series elastic actuators. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 399–406. https://doi.org/10.1109/IROS.1995.525827
3. Isermann R (2011) Mechatronic systems: fundamentals. Springer. https://doi.org/10.1007/1-84628-259-4
4. Goodwin GC, Graebe SF, Salgado ME (2001) Control system design. Prentice Hall. https://books.google.com/books?id=7dNSAAAAMAAJ
5. Åstrom KJ, Wittenmark B (1997) Computer-controlled systems: theory and design. Prentice Hall. https://books.google.com/books?id=5Z0QAQAAMAAJ
6. Armstrong-Hélouvry B, Dupont P, Wit CC (1994) A survey of models, analysis tools and compensation methods for the control of machines with friction. Automatica 30(7):1083–1138. https://doi.org/10.1016/0005-1098(94)90209-7
7. Canudas-de-Wit C, Olsson H, Åstrom KJ, Lischinsky P (1995) A new model for control of systems with friction. IEEE Trans Autom Control 40(3):419–425. https://doi.org/10.1109/9.376053
8. Vanderborght B, Albu-Schaeffer A, Bicchi A, Burdet E, Caldwell DG, Carloni R, Catalano M, Eiberger O, Friedl W, Ganesh G, Garabini M, Grebenstein M, Grioli G, Haddadin S, Hoppner H, Jafari A, Laffranchi M, Lefeber D, Petit F, Stramigioli S, Tsagarakis N, Van Damme M, Van Ham R, Visser LC, Wolf S (2013) Variable impedance actuators: a review. Robot Auton Syst 61(12):1601–1614. https://doi.org/10.1016/j.robot.2013.06.009
9. Buttazzo GC (2011) Hard real-time computing systems. Springer. https://doi.org/10.1007/978-1-4614-0676-1
10. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J Acm 20(1):46–61. https://doi.org/10.1145/321738.321743
11. Blockwitz T, Otter M, Arnold M et al. (2012) Functional mockup interface 2.0: the standard for tool independent exchange of simulation models. In Proceedings of the 9th International Modelica Conference, pp 173–184. https://doi.org/10.3384/ecp12076173
12. Reynaud J, Enfrad B, Lefrançois A (2025) Facilitating the use of physics-based simulations on embedded devices by running fmus from micropython. In Proceedings of the 16th International Modelica & FMI Conference. https://doi.org/10.3384/ecp218535
13. IEC: IEC 61131–3 (2013) Programmable controllers – programming languages. IEC Standard. https://webstore.iec.ch/en/publication/4552
14. Kritzinger W, Karner M, Traar G, Henjes J, Sihn W (2018) Digital twin in manufacturing: a categorical literature review and classification. IFAC-Papersonline 51(11):1016–1022. https://doi.org/10.1016/j.ifacol.2018.08.474
15. Tao F, Zhang M, Liu A, Nee AYC (2019) Digital twin in industry: state-of-the-art. IEEE Trans Ind Inf 15(4):2405–2415. https://doi.org/10.1109/TII.2018.2873186
16. Cao JEA (2025) Construction and efficiency analysis of an embedded system-based verification platform for edge computing. Sci Rep 15(1). https://doi.org/10.1038/s41598-025-10580-3
17. Mikelsons L, Samlaus T (2017) Towards virtual validation of ecu software using fmi. In Proceedings of the 12th International Modelica Conference, pp 307–316. https://doi.org/10.3384/ecp17132307
18. IEC: IEC 61508 (2010) Functional safety of electrical/electronic/programmable electronic safety-related systems. IEC Standard. https://webstore.iec.ch/en/publication/5515
19. ISO: ISO 26262 (2018) Road vehicles – functional safety. ISO Standard. https://www.iso.org/standard/68383.html
20. Modelica Association (2014) Functional mock-up interface for model exchange and co-simulation, version 2.0. https://fmi-standard.org/docs/2.0/
21. Gomes C, Thule C, Broman D, Larsen PG, Vangheluwe H (2018) Co-simulation: a survey. ACM Comput Surv 51(3):49. https://doi.org/10.1145/3179993
22. Broman D, Brooks C, Greenberg L, Lee EA, Masin M, Tripakis S, Wetter M (2013) Determinate composition of fmus for co-simulation. In Proc. EMSOFT. https://doi.org/10.1109/EMSOFT.2013.6658580
23. Modelica Association (2022) Functional mock-up interface for model exchange and co-simulation, version 3.0. https://fmi-standard.org/docs/3.0/
24. Hansen ST, Gomes CNG, Najafi M, Sommer T, Blesken M, Zacharias I, Kotte O, Mai PR, Schuch K, Wernersson K, Bertsch C, Blochwitz T, Junghanns A (2022) The fmi 3.0 standard interface for clocked and scheduled simulations. Electronics 11(21):3635. https://doi.org/10.3390/electronics11213635
25. Lenord O et al. (2021) efmi: embedded functional mock-up interface for deterministic code generation. In Proceedings of the Modelica Conference. https://doi.org/10.3384/ecp2118157
26. Armugham SS et al. (2021) Integrating eFMI-based models into AUTOSAR-compliant embedded systems. SAE Technical Paper. https://doi.org/10.4271/2021-26-0048
27. Ultsch J, Ruggaber J, Pfeiffer A, Schreppel C, Tobolář J, Brembeck J, Baumgartner D (2021) Advanced controller development based on efmi with applications to automotive vertical dynamics control. Actuators 10(11):301. https://doi.org/10.3390/act10110301
28. Arias GAB, Porto AJV (2022) Simulation of complex actuator of a large agricultural robot for deep reinforcement learning. In Congresso Brasileiro de Automática (CBA). https://doi.org/10.20906/CBA2022/3210
29. Lepais C, Zimmer D (2025) Prototypical control for the digital twin of aircraft environmental control system. In Proceedings of the 16th International Modelica & FMI Conference, pp 81–90). https://doi.org/10.3384/ecp21881
30. Oldemeyer U et al. (2024) MoRC: a modular robot controller based on FMI. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). https://doi.org/10.1109/ICRA57147.2024.10611553 WoS: WOS:001369728006060
31. Klemets K, Bouzoulas D, Manngård M, Böling JM (2026) FMUiL: an open-source package for in-the-loop simulations with FMUs. SoftwareX 34:102560. https://doi.org/10.1016/j.softx.2026.102560. WoS: WOS:001698039600001
32. Page MJ, McKenzie JE, Bossuyt PM et al. (2021) The prisma 2020 statement: an updated guideline for reporting systematic reviews. BMJ 372(71). https://doi.org/10.1136/bmj.n71

33. Modelica Association (2025) FMI 3.0 implementers' guide (living document). https://modelica.github.io/fmi-guides/main/fmi-guide/
34. Modelica Association (2026) FMI standard – functional mock-up interface. https://fmi-standard.org
35. Modelica Association (2025) FMI tools and projects. https://fmi-standard.org/tools/
36. Dassault Systèmes (2026) Dymola – multi-engineering modeling and simulation. https://www.3ds.com/products/catia/dymola
37. Siemens Digital Industries Software (2026) Simcenter amesim – 1D system simulation. https://plm.sw.siemens.com/en-US/simcenter/systems-simulation/amesim/
38. Maplesoft (2026) MapleSim – physical modeling and simulation. https://www.maplesoft.com/products/maplesim/
39. ESI Group (2026) SimulationX – physical system simulation. https://www.simulationx.com
40. Modelon (2026) Modelon impact. https://modelon.com/modelon-impact/
41. Orthogonal Inc. (2026) Orthogonal ODE – Modelica + AI MBSE platform. https://www.orthogonal.dev
42. Ansys Inc. (2026) ANSYS twin builder. https://www.ansys.com/products/digital-twin/ansys-twin-builder
43. Claytex (2026) FMI Blockset for Simulink. https://www.claytex.com/tools/fmi-blockset/
44. Open Source Modelica Consortium (2026) OpenModelica – open source modeling and simulation environment. https://openmodelica.org/
45. Modelica Association (2024) Functional Mock-up interface (FMI) specification 3.0.2. https://fmi-standard.org/docs/3.0.2/
46. Modelica Association (2024) FMI 3.0.1 release announcement (MA-20-11027). https://fmi-standard.org/docs/3.0.1/
47. AIT Austrian Institute of Technology (2026) FMITerminalBlock – FMI to IEC 61499 bridge. https://github.com/AIT-IES/fmi-terminal-block
48. Bosch Research (2026) fmi_adapter – FMI Adapter for ROS 2. https://github.com/boschresearch/fmi_adapter
49. Walther K, Wetter M, Prakash A, Hu J (2025) Cdl-plc translator: from modelica hvac control design to iec 61131 plc implementation. In Proceedings of the 16th International Modelica & FMI Conference, pp 515–524). https://doi.org/10.3384/ecp218515
50. Khalil HK (2002) Nonlinear systems. Prentice Hall. https://flyingv.ucsd.edu/krstic/files/Khalil-3rd.pdf
51. Hogan N (1985) Impedance control: an approach to manipulation: part I—theory. J Dyn Syst Meas Control 107(1):1–7. https://doi.org/10.1115/1.3140702
52. robotology community (2026) Gazebo-fmi Plugin for gazebo. https://github.com/robotology/gazebo-fmi
53. Lange J, Stogl D, Matheis D (2025) Towards native fmi support in ros–gazebo pipelines. In Proc. 16th International Modelica & FMI Conference, pp 325–334. https://doi.org/10.1007/978-3-030-45956-7-7
54. Rindaroy M, Nordahl H, Sadjina S, Skjong S, Hagaseth M (2025) Adding higher-level semantics to Functional Mock-up units for easier, faster, and more robust co-simulation connections. Softw Syst Model 24(2):471–487. https://doi.org/10.1007/s10270-024-01244-3. WoS: WOS:001407457500001
55. Serban R, Mangoni D (2026) Functional mock-up interface support in Chrono. Multibody Syst Dyn 66(2):295–311. https://doi.org/10.1007/s11044-025-10106-9. WoS: WOS:001567660400001
56. Perabo F, Zadeh M (2025) Multiphysics modeling and co-simulation of ship electric power and propulsion systems for virtual testing and verification. IEEE Trans Transp Electrific 11(1):5108–5121. https://doi.org/10.1109/TTE.2024.3480009. WoS: WOS:001416302800013
57. Silva LTFW, Tomim MA, Barbosa PG, de Almeida PM, da Silva Dias RF (2023) Modeling and simulating wind energy generation systems by means of co-simulation techniques. Energies 16(19):7013. https://doi.org/10.3390/en16197013. WoS: WOS:001085582800001
58. Eklund S, Falk M, Palmkvist G (2021) Energy optimization of rail vehicles using fmi-based virtual drives. In Proceedings of the Modelica Conference. https://doi.org/10.1016/j.tra.2003.07.001
59. Picerno M, Lee S-Y, Pasternak M, Siddareddy R, Franken T, Mauss F, Andert J (2021) Real-time emission prediction with detailed chemistry under transient conditions for hardware-in-the-loop simulations. Energies 15(1):261. https://doi.org/10.3390/en15010261. WoS: WOS:000750891500001
60. Ultsch J, Ahmic K, Brembeck J (2025) Train_fmu_gym: a Functional mock-up unit-based framework to train reinforcement learning agents for multi-physical systems. IEEE Access 13:208845–208867. https://doi.org/10.1109/ACCESS.2025.3636787. WoS: WOS:001641511700048
61. Rojas-Ordoñez S, Segura M, Zulueta E (2025) Integration of physical and ai models using open and interoperable standards: a model-based methodology for autonomous robot development. In Proceedings of the 16th International Modelica&FMI Conference, September 8–10, 2025, Lucerne University of Applied Sciences and Arts (HSLU). Linköping Electronic Conference Proceedings, vol 218. pp 479–488. https://doi.org/10.3384/ecp2188479
62. Petridis K, Clauss C (2015) Test of basic co-simulation algorithms using fmi. In Modelica Conference. https://doi.org/10.3384/ecp15118865
63. Nicolai A et al. (2019) Robust and accurate co-simulation master algorithms applied to fmi slaves with discontinuous signals using fmi 2.0 features. In Modelica Conference. https://doi.org/10.3384/ecp19157769
64. Audsley NC (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. https://www.cs.york.ac.uk/rts/publications/Audsley1991a.html
65. Modelica Association (2022) What's New in FMI 3.0 – summary for users and tool vendors. In Presentado en NAFEMS European Conference 2022. https://nafems.org/downloads/24658/
66. Modelica Association Project (2021) eFMI Overview – embedded functional mock-up interface. In NAFEMS European Conference on Multiphysics Simulation, 2021. https://nafems.org/downloads/24032/
67. U.S. Department of Energy (2019) Emphysis final public report. Technical report, Modelica Association Project https://www.modelica.org/projects/emphysis/EMPHYSIS-Project-Public-Report.pdf
68. Modelon and Åbo Akademi (2026) FMI Library – C implementation of the FMI standard. https://github.com/modelon-community/fmi-library
69. Dassault Systèmes and Contributors (2026) Fmpy – python Library for simulating FMUs. https://github.com/CATIA-Systems/FMPy
70. Modelon (2026) PyFMI – python package for working with FMUs. https://github.com/modelon-community/PyFMI
71. Modelica Association (2026) FMI cross-check rules and validation. https://fmi-standard.org/cross-check/
72. prostep ivip (2023) Public SSB fact sheet: functional mockup interface (FMI) – includes FMI cross check rules. Fact sheet. https://www.prostep.org/fileadmin/fact-sheets/Public_SSB_Fact_Sheet__Functional_Mockup_Interface__FMI_-v2-20231219_081833.pdf
73. Modelica Association (2026) FMI cross-check repository. https://github.com/modelica/fmi-cross-check
74. Modelon, Åbo Akademi (2026) FMI Library – open source C library for FMI integration. https://github.com/modelon-community/fmi-library

75. RTCA: DO-178C (2011) Software considerations in airborne systems and equipment certification. RTCA Standard. https://my.rtca.org/nc__store?category=a0S3600000DWOZ6
76. EMPHYSIS Consortium (2024) Emphysis public demonstrator summary / public report on efmi-related outcomes (deliverable d7.9). Technical report, EMPHYSIS Consortium https://www.efmi-standard.org/wp-content/uploads/2024/05/emphysis-public-demonstrator-summary.pdf,
77. Modelica Association Project eFMI (2024) Efmi annual project report 2024. Technical report, Modelica Association Project eFMI https://www.efmi-standard.org/media/resources/MAP-eFMI-annual-project-report-2024.pdf,
78. MISRA (2012) MISRA C: 2012 (guidelines / publication information). MISRA. https://www.misra.org.uk/
79. Wolf C, Schleipen M, Frey G (2023) Secure exchange of black-box simulation models using fmi in the industrial context. In Proceedings of the Modelica Conference. https://doi.org/10.3384/ecp204487
80. MathWorks (Simulink Blog) (2019) Fmus for model exchange: are FMUs a good way to hide IP? https://blogs.mathworks.com/simulink/2019/05/16/fmus-for-model-exchange/
81. UniFMU authors. (2025). Fmi-based distributed co-simulation with enhanced security and intellectual property safeguards. arXiv preprint (https://doi.org/10.48550/arXiv.2510.20403
82. Modelica Association (2024) FMI validation resources and updates (incl. Cross-check / validation tooling references). https://fmi-standard.org/
83. FMU Compliance Checker (Open-source tool) (2024) FMU compliance / validation tool referenced by FMI validation resources. Software repository. https://fmi-standard.org/

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.