Barriers for Adopting FMI-based Co-Simulation in Industrial MBSE Processes [*]

Johan Cederbladh, Anna Reale, Andreas Bergsten, Richard Mikelöv, Antonio Cicchetti

johan.cederbladh@mdu.se

October 16, 2023

Abstract

Model-Based Systems Engineering (MBSE) is a growing paradigm for system development where models are the primary considered artefacts. However, MBSE often relies on semi-formal modelling languages and methods, limiting analytical capabilities. Co-Simulation is argued in the literature to be a promising technology in the simulation domain for integrating heterogeneous models in unified simulations. The most commonly used standard for Co-Simulation is currently the Functional-Mockup-Interface (FMI), supported by many tools in the industry. Recently there has been increasing interest in utilizing co-simulation in MBSE processes to enable simulation capabilities earlier in development, mainly via instantiating simulations using the FMI standard from system architecture views. This paper briefly argues the case for co-simulation for industrial MBSE and presents several barriers to integration from a holistic point of view. The paper highlights the need for further research and progress to improve the maturity of the industrial adoption for MBSE workflows while discussing the current outlook for FMI-based co-simulation orchestrated from architecture models.

Keywords: MBSE, Co-simulation, FMI, SysML, Systems Engineering

1 Introduction

Systems Engineering (SE) is a discipline for managing systems throughout their life-cycle. The International Council On Systems Engineering (INCOSE) broadly define a system as: *"... an integrated set of elements, subsystems, or assemblies that accomplish a defined objective."* [34]. Indeed, using SE is

[*]This is a pre-print for an article accepted at a MODELS2023 workshop. This work was partly funded by the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement No. 101007350.

often motivated by the need to manage complexity and introduce rigorous processes for the system life-cycle. Currently, there is a rise of complexity in the systems developed, partly it can be attributed to the rise of software [24], and the increasingly heterogeneous domain integration, for example, Cyber-Physical Systems (CPSs) [30]. SE is undergoing a digital transformation to meet these challenges, including adopting surrounding technologies to meet the increasing demands of system design and development, such as DevOps [11]. However, one of the more prominent evolutions of SE is using models as the underlying core artefacts, also referred to as Model-Based Systems Engineering (MBSE) [23,35]. In this case, a model is considered to describe something without being the thing itself [21]. Since the use of models relies on abstraction, it is essential to have a clear purpose and scope of a model, as it often is only valuable for a given specific context. In the domain of MBSE, the *de facto* standard language for system models is SysML from the Object Management Group (OMG)[1]. SysML is a semi-formal language, originally extended from the UML, that uses various diagrammatic *views* to represent a system. Models are seen as a powerful means of tackling complex systems descriptions while adding value, such as increased traceability and collaboration [12]. At this stage, the use of models, and in particular SysML, is a best industry practice. More recently, there has been an increasing wish to leverage SysML models, often created at *early* stages of development (and maintained throughout), for enhanced analysis to further leverage the benefits of digital models and provide businesses with a competitive edge during development.

SysML models are often not inherently executable due to the semi-formal nature and the often high-level abstraction in models. In order to use SysML models for *meaningful* simulation, they often require more rigour, which can be done through various techniques from model-based engineering, like model transformations, or more recently by leveraging Co-simulation [13, 27]. Co-Simulation is a technique that leverages some form of interchange between simulation models, enabling more modular and flexible simulation, for example, using different notations in a uniform simulation. Currently, the most widespread standard for co-simulation is the Functional-Mockup-Interface (FMI)[2], which is supported by a wide array of tools. The FMI standard, in particular, enables co-simulation via abstracting simulation models to black boxes, which only detail input and output signals and possibly some model parameters. Engineers can then freely connect these models to create system simulations regardless of the underlying notation and tools for the individual models.

One potential use of co-simulation when leveraging the FMI standard is using SysML diagrams as architecture descriptions, which then are mapped to simulation models to orchestrate system simulations. This way, a user can use SysML to define system architectures which can re-use simulation models compatible with co-simulation enabling simulation orchestration from SysML without the need for detailed simulation knowledge, potentially lifting the simulation capa-

[1]https://www.omg.org/spec/SysML
[2]https://fmi-standard.org/

bility *earlier* in the design process for a wider audience. However, integrating co-simulation technologies in MBSE processes is more complex than often portrayed in literature. This can be attributed to the use of co-simulation originally not being aimed at MBSE processes, even though there is increasing industrial attention. In this paper, we discuss some practical inhibitors of co-simulation and MBSE, and we use the SysML language and FMI standard as the context for co-simulation and present our views on what steps need to be taken going forward.

The rest of the paper is structured as follows, Section 2 further provides the necessary background and related work on crucial terminology and topics. Section 3 gives an overview of industrial motivation. In Section 4, we present and discuss our identified barriers. Section 5 provides a discussion of the paper, and finally, Section 6 concludes the paper and presents future work.

2 Background and related work

SE is a discipline which regards standard practices and processes for development and often guiding standards such as ISO 15288[3] and ISO 42010[4] which detail general approaches for the development of systems are used to guide engineers. Notably, the standard procedure is development in *stages* using various system *views* to describe systems. There are many examples of workflows across development stages created for MBSE in literature [9], and often they might be particular for a specific domain. SE best-practices often aim to reduce risks for faulty design, as the cost of addressing a design issue increases exponentially as the system development progresses [34]. Nevertheless, SE is often an iterative approach, which takes customer needs and requirements to express a *problem*, where the aim is to find a suitable *solution*, often based on some trade-offs for the particular case. Indeed, SE is often a process of systematically narrowing design through various stages during development, such as the standard V-model [34]. In the early stages of development, when systems are described in high-abstraction and using functional and logical descriptions, the solution space will often be extensive and practically endless regarding large product families and systems with high variability [6]. In this way, iterative processes are often needed to continuously shrink the solution space based on analysis and refined customer requirements and needs. Part of the process is gradually limiting the design by introducing stricter requirements or locking architectural design. These processes eventually lead to engineers making decisions that will determine the detailed design.

With the digitalisation paradigm shift, MBSE is seeing increasing industrial adoption [14, 32, 33]. MBSE replaces the traditionally document-centric development by employing models at all stages of development as the primary artefacts of development. Practically, MBSE is often related to using semi-formal languages like SysML for system descriptions [12]. As the standard practice for

[3]https://www.iso.org/standard/63711.html
[4]https://www.iso.org/standard/74393.html

MBSE involves SysML, there are limitations to what can be analysed at the language level. The strength of semi-formal languages like SysML is the freedom of extension and usage, along with often mature tooling integrated capabilities of traceability and allocation of stakeholder concerns [8]. However, the literature argues that simulation is a necessary capability of SysML [27]. Strengthened analytical capabilities could improve decision-making while increasing the overall value of employing MBSE [1]. Typically simulation will not be done on SysML models directly, and there is extensive reporting on the use of model transformations [28].

More recently, extending SysML models with support for FMI instantiated simulation has been introduced [3, 13, 27]. From a high-level view, the FMI standard describes a format for black box representations of simulation models called Functional-Mockup-Units (FMUs). The considered simulation models are compiled into inaccessible source code only detailing inputs, outputs, and some model parameters (more so in the newer 3.0 version). FMUs can be seen as simulation "building blocks", where these models have a high degree of abstraction, which can be connected via their interfaces (inputs - outputs) to create more extensive simulations. A key benefit in this regard is the ability to reuse and combine models defined in different languages and tools. Additionally, the abstraction introduced by FMI maps well to typical SysML models, paving the way for the linkage of models using automated means—particularly block diagrams map well to the FMI abstraction layer.

However, previous industrial reports on FMI have been critical of the standard functionality [5], for example, due to the non-determinism and wrongful output for even simple cases like a bouncing ball. Similarly, academic reporting has been done on the FMI standard, detailing many issues that significantly impact the standard's useability [29]. The newer version of FMI 3.0 promises to improve many of the observed issues, but it remains to be seen whether that can be confirmed on a broader scale [20]. In this paper, we provide a similar report from the context of industrial MBSE processes and highlight our view on current inhibitors for co-simulation integration based on our experiences with both the FMI standard and MBSE relying primarily on SysML.

3 Industry motivation and case

As mentioned, industrial SE is predominantly based on standard processes and methods. Typically a system life-cycle is divided into stages, considering *gates*, which mark distinct transitions between life-cycle stages. ISO 15288 is a standard that exemplifies a generic system life cycle process with several stages, Concept, Development, Production, Utilisation/Support, and Retirement. There are many similar definitions and flows, for example, detailed in the INCOSE SE handbook [34]. As the system traverses the life-cycle, the knowledge and confidence of the system increases, manifested as increasing detail during design and operational data once the system is deployed. In addition to system knowledge and confidence increasing, there are typically different people and roles with corresponding responsibilities at distinct stages. For example, a Requirements engineer, system architect, and system verification engineer will typically consider different views of a system. The need for unification in development and between teams is increasing with the rising complexity of systems. Models and MBSE is a means of connecting different stakeholders across developmental stages in more integrated views. The rise of MBSE can partly be attributed to the rise in digitalization, and for SE the notion of a systems *digital thread* is a growing domain [31]. The digital thread concept and parallel technical advancements aim to create robust traceability and data sharing across traditionally separate *silos* of development. Using models (or other digital artefacts) could promote collaborative efforts by leveraging abstraction with common languages and tools.

In SE, it is essential to re-use knowledge, and often previous analysis results can be used as proof of viable design [34]. Typically service history of a particular system can impact the design of a new version or re-use of subparts of a product to avoid redoing expensive verification or validation tasks. Co-simulation via FMI (or other standards) promises a new method of re-use, allowing models to be seen as abstract blocks that can be used for analysis earlier than traditional means of simulation. In particular, the black box view maps well to the internal block diagrams, which typically describe system architectures in languages like SysML. A high-level view of how the FMI standard might typically be implemented is visualised in Figure 1. Based on a SysML model, information is extracted to instantiate a simulation configuration. Typically a Simulation will consist of a System under Study (SuS), a Scenario, and some Evaluation. A generic approach to extracting this data from typical MBSE models is by interpreting block diagrams as the SuS and identifying appropriate simulation models that match the SuS from a library of FMUs. A benefit of considering co-simulation formats as FMI is that the simulation models do not necessarily need to originate from the same language or tool. As such, flexibility is introduced in the simulation context as models can be picked more freely in a modular fashion to create simulation models. The Scenario will consider the system's environment and any external factors that provide input for the experiment at hand, typically read from parameter representations in the SysML model. The evaluation part of the simulation is required to interpret the results. It is typically based on requirements from the original model to define what should be observed and what constitutes a pass or fail in the experiment.

Leveraging model-based methods increases opportunities to create unified system models that can act as single sources of truth. Introducing a SysML model as a "back-bone," engineers can leverage the model as an information source to create, for example, simulations as part of the analysis, particularly in the early stages of development. Using SysML further enables users access to domain-specific simulation models previously not accessible by a wider audience due to tooling, notation, and simulation knowledge. Simulation at early stages is most of the time used for initial virtual experiments or "what-if" scenario analysis. The system is limited to the SysML model, so it can often be very coarse-grained. Introducing this capability could, as such, be a means of

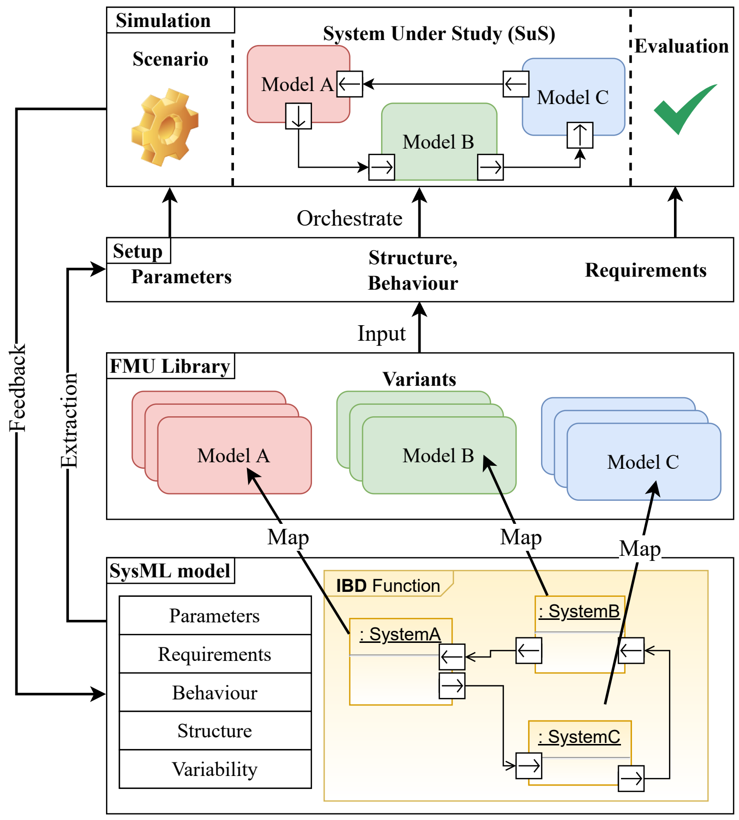

Figure 1: Leveraging MBSE and FMI for increased digital workflows.

making more informed decisions based on the traditionally used SysML models, or decisions could perhaps be made *earlier* compared to traditional processes. Adding automation to the described process would create value, as legacy silos often need to manually transfer domain knowledge between one another. As such, the case for introducing co-simulation processes in MBSE is seemingly worth pursuing. An added benefit of an early definition of a single source of truth is that it can be maintained throughout the project, reducing the need for additional tools and maintaining a more robust unification of artefacts through a digital thread.

4 Adoption barriers

Based on the previous sections, there is substantial motivation and value proposition for introducing co-simulation in MBSE settings. This section details what the authors believe are current barriers to co-simulation (using FMI) in industrial MBSE settings (using SysML) based on experiences of implementation. Each identified barrier will be discussed regarding the perceived problems and potential solutions, summarized at the end of each sub-section with highlights for future research.

4.1 Integration in SE workflows

SysML models are typically used for systems engineers to design a system top-down, breaking down requirements and providing adequate allocation through the system design. Simulation as an analysis method requires a certain level of maturity in system descriptions, at least when used for Validation and Verification (V&V) purposes. In traditional SE processes, these activities are performed in distinct stages of development. As such, it begs the question, who does the co-simulation capability cater to? A system engineer typically does not need to consider simulation analysis, as the engineer will traditionally be concerned with the system from a logical perspective and focus on requirements, traceability, etc. At the same time, simulation engineers already have access to advanced simulation environments and the corresponding knowledge to leverage these tools for complex analysis. Considering this, there is a need to enable a smooth integration between the various involved actors and considered domains. In the digital thread, seamless integration between various developmental efforts is needed, and engineers need to have knowledge of both domains for integration purposes, visualized in Figure 2.

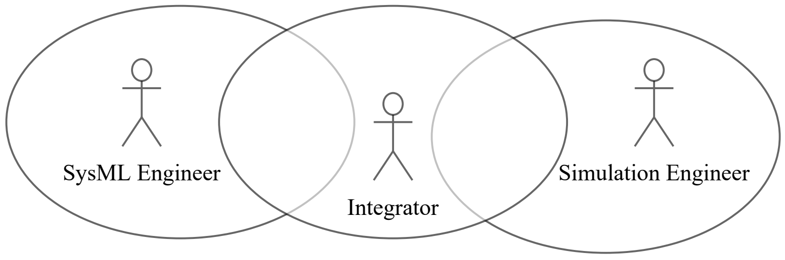

Figure 2: Stakeholders of MBSE co-simulation.

While the `Integrator` in the figure could refer to partly tooling-related concepts, it begs the question of *who* these stakeholders would be. Additionally, the notion of *why* you should perform co-simulation is similarly essential to who has the primary stake in the process. The application of these technologies should improve the processes for product development and design. In this case, it must be clear that the capability provides something new in value compared to existing methods. The two other stakeholders in this case, the `SysML` and `Simulation` engineers should be involved with the `Integrator`. However, in this case, there is a clear gap in the traditional responsibilities and knowledge, so effort and increased resources must be allocated to bridge the domains.

Furthermore, employing co-simulation with SysML for architecture definition requires a mature understanding of how the mapping should be made between the two notations. At this point, the required tooling and techniques are realized by various practical means, for example, model transformations. How-

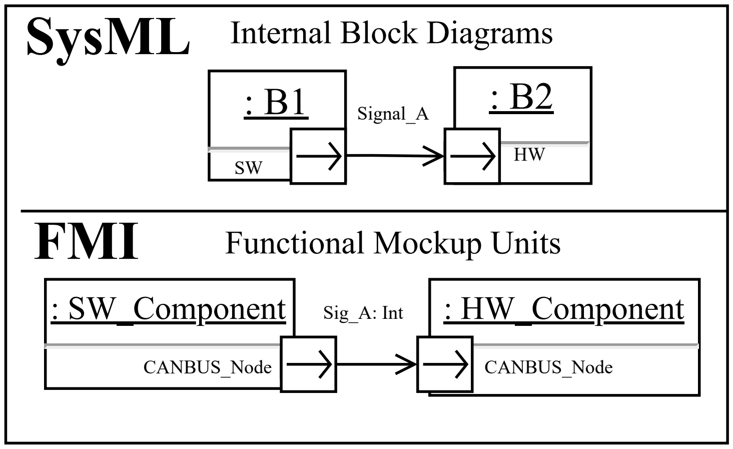

Figure 3: A typical gap between SysML diagrams and FMUs for co-simulation.

ever, there is a gap in a systematic means of guiding a system modeller to portray *enough* information for cross-domain simulation. Often system modelling regards logical descriptions of systems, and many typical approaches for MBSE, such as MagicGrid, regard physical properties as not part of MBSE [26]. In this way, often *logical* architectures are defined, which for example, only considers that there *is* a signal and not *what* the signal is. Figure 3 highlights a typical instance of the gap, where SysML models with logical architectures should be mapped to FMUs with correspondingly higher levels of detail. The figure shows a typical manifestation of the gap, as the SysML model often highlights the presence of different blocks with corresponding interfaces typed, such as HW or SW with a standard signal A. For physical simulation, the same components in the FMI format could, for example, explicitly mention the communication protocol, signal types, and communication speed to enable simulation.

The standard means of using MBSE is not directly compatible with co-simulation since physical information is often missing or represented in ad-hoc fashions. Physical simulation models defined in languages such as Simulink or Modelica contain more physical information than SysML models, which has traditionally not been an issue due to the underlying difference in technical audiences. With the advent of co-simulation there is an expected merge of concerns. Still, the fact remains that SysML block diagrams typically only consider logical signals rather than physical ones, as it is simply not of interest to a system modeller. While the solution often boils down to the need to increase detail in the SysML model, perhaps with another view or with a corresponding stereotype, it still poses the challenge of implementing those details as that is typically not the responsibility of a system engineer/architect. This is a knowledge gap, and work is required to understand better how this gap can be bridged with re-use, collaboration, or reasoning. While mechanisms to extend the SysML language exist, and there are many examples of successful implementation, the standard methods of implementing SysML omit the notion of simulation integration. Without a more unified view from the community, there is a considerable risk that the successful integration will remain anecdotal and ad hoc. In particular, referring to Figure 3, the typical "end" for MBSE approaches does not detail the physical properties necessary for simulation.

So while the notion of using co-simulation with a technical integration from SysML and FMI is technically sound, which engineer it caters to in the developmental process needs to be clarified. Co-simulation is one of many proposed capabilities for MBSE, particularly in the move towards seamless integration between development stages. While a more integrated workflow is attractive, it necessitates that the traditional engineering roles are revisited and that new methods and techniques are adequately satisfying those engineering needs. This is a challenge regarding the use of co-simulation, particularly as the use of Co-simulation in industrial MBSE flows has increased potential value proportional to how *early* the considered system is in its developmental cycle. The ability to simulate systems for early analysis already by reading requirements and orchestrating *valid* simulations could significantly boost developmental progress and company competitiveness.

However, parallel considerations of uncertainty arise by introducing the notion of the early phase or stage. A system at the early stages of development is not fully understood, and details are often not yet realized or to be determined with the corresponding analysis. Therefore the ability for system simulation should be considered carefully as it most likely is re-using simulation models from other systems or contexts, requiring some analysis on the feasibility and validity of re-using those models. At the same time, due to the presence of uncertainty at this stage, some assumptions of the system and its environment are most likely needed, further hindering the ability of reasoning about simulation validity. It might be possible to re-use knowledge about previous cases and analysis. However, it requires mature methods of storing the necessary information accessible and scaleable, somewhat hindered by the black box representation of the eventual simulation models.

Highlight and summary

Mapping the SysML language to the FMI standard is at this point well known. However, it is less clear who the technology caters to in the scope of the broader SE landscape. Additionally, the notion of early analysis is not well understood, and it is necessary to evaluate how high-fidelity simulations can guide early decision-making in low-fidelity system models.

4.2 Leveraging FMI for simulation

FMI is positioned as a critical enabler for interconnected simulations using heterogeneous models, leveraging black box representations to hide the underlying model complexity and implementation. There are reported examples of successful integration in the industry [15, 18], and actors as Bosch praising it as being a key enabler in the future[5] (particularly the newer 3.0 version). Indeed, the need and willingness for the industry to adopt co-simulation through FMI is increasing in recent times. Notably, the notion of model re-use and reduction of vendor lock-in pushes adoption.

However, as mentioned in the background, version 2 of FMI has been studied in academia and industry for its issues during simulation. Particularly the instability of simulation and non-determinism have been technical barriers necessary to overcome. The newer 3.0 version promises to reduce a lot of the existing issues. As part of ongoing work, we evaluate the reliability of FMI models and extract a small simulation case study from the work [25]. Models compliant with FMI under various setups in commonly used tools, namely Simulink and OpenModelica, are simulated. Results are compared between baseline simulations (in the corresponding tool) and the same simulation with FMI formatted models (again simulated in the same tool). To promote reproducibility, we utilize a Mathworks example model of a vehicle model[6] as the SuS. The model is constructed at a glance, as seen in Figure 4.

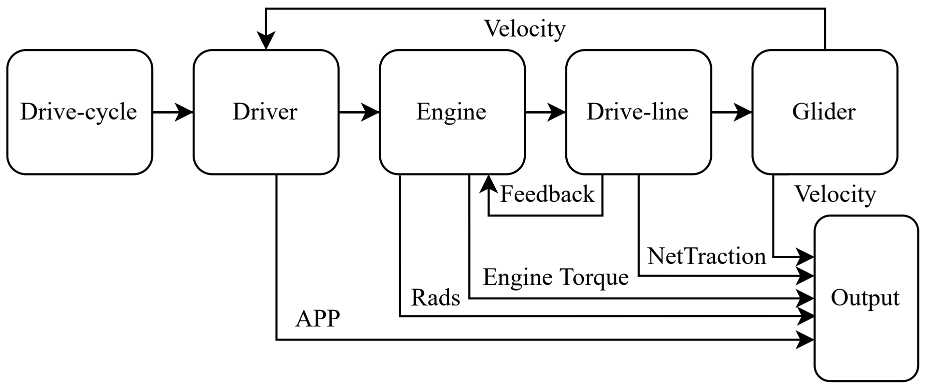

Figure 4: High-level view of the vehicle simulation.

The model is a holistic simulation of a vehicle stimulated with a particular input drive-cycle. It consists of five major sub-systems, the `Drive-cycle`, `Driver`, `Engine`, `Drive-line`, and `Glider`. A drive-cycle stimulates the driver, and the rest of the blocks are connected in a feedback loop. For our evaluation, we take five signals as outputs, Accelerator Pedal Position `APP`, Rotational speed of the engine `Rads`, `Engine Torque`, Net Tractive Force `NetTraction`, and `Velocity`. At least one signal from each block is viewed and logged for execution traces to analyze. We perform a rudimentary evaluation of the standard stability by re-employing the individual model elements as FMUs and orchestrating simulation via co-simulation. The results highlight numerical issues with both FMI 2 and FMI 3 with "off the shelf" solutions. Additionally, we use a

[5]https://www.bosch.com/stories/fmi-3-0-the-next-generation-exchange-format-for-system-simulation-beyond-tool-borders/

[6]https://github.com/mathworks/vehicle-modeling/releases/tag/v4.1.1

standard drive cycle[7] (class 3b) as the primary input for the driver model. We utilize two versions of Simulink (2023a and 2022b) to create and simulate the vehicle simulation models. We then export the models to FMUs using Simulink functionality and again execute them in the Simulink or OpenModelica tools and compare the execution traces. In the case of FMI version 3, we only do it for Simulink. Table 1 shows a high-level comparison view where we detail the previously mentioned monitored signals error in execution between FMU execution and the original model in terms of max, average, and duration of errors. We note that in this particular case, we are not interested in the viability of the original model but rather in the reliable reproduction of execution of the exported FMUs. The export function is made using the standard procedure of the tools and validated based on available methods from the FMI project[8].

Table 1: Comparison of FMI execution and original model execution

FMI version 2 execution (OpenModelica) based on Baseline23a			
Signal	**Max Error**	**Average Error**	**Error Duration**
Torque [Nm]	236.85974	4.74922	33.94035%
Rads [Rad/s]	144.59612	3.78094	99.99946%
NetTractivForce [N]	2133.16059	23.55839	35.14792%
Velocity [Km/h]	11.68082	0.988475	99.99946%
App [%]	64.18139	2.830221	34.28738%
FMI version 2 execution (Simulink) based on Baseline23a			
Torque [Nm]	101.38549	0.50454	33.69225%
Rads [Rad/s]	81.85649	0.78449	99.99946%
NetTractivForce [N]	2695.53942	9.55547	34.82305%
Velocity [Km/h]	0.33310	0.01355	99.99946%
App [%]	14.87628	0.14465	34.04036%
FMI version 3 execution (Simulink) based on Baseline22b			
Torque [Nm]	46.21097	0.15584	33.67495%
Rads [Rad/s]	81.31894	0.11835	99.99783%
NetTractiveForce [N]	2190.38639	2.28989	34.19549%
Velocity [Km/h]	0.14663	0.00398	99.99946%
App [%]	6.970128	0.04858	34.00251%
FMI version 3 execution (Simulink) based on Baseline23a			
Torque [Nm]	82.04390	0.44762	33.69333%
Rads [Rad/s]	81.16435	0.75782	99.99946%
NetTractivForce [N]	2697.39773	8.75499	34.79548%
Velocity [km/h]	0.31593	0.01233	99.99946%
App [%]	14.75038	0.12734	34.04305%

The table summarises exporting individual sub-systems, simulating via Co-simulation, and comparing the results to the original Simulink execution traces

[7]https://www.wltpfacts.eu/what-is-wltp-how-will-it-work/

[8]https://fmi-standard.org/validation/

from two versions at each time step. We average the discrepancy over time, sampled 100 times per second in fixed step size over the standard drive-cycle (30 min). In both tables, the execution traces differ from the original model. While it might be hard to position the effect of the error from the table alone, it is a noticeable issue. Perhaps more alarming, the simulation runs differ significantly for subsequent standard versions of Simulink (used to export the models as FMUs). For some attributes, the signal errors are constant, for example, `Rads` and `Velocity`, which is not strange due to the carry-over effect of the errors. The `NetTractiveForce` instead is primarily correct (within a slight error of margin) but has more significant spikes of errors (around size 50% of total signal value for each simulation), we refer to [25] for a more extensive evaluation. While the example is relatively simple and can be studied more extensively, it highlights the ambiguous nature of the results and simply changing versions of the same tool. The models can be more complex in larger system models, and in larger SE contexts, there could most likely be a more diverse tool and language landscape. Although the tables present a limited view regarding averaging the error and not detailing the overall drive-cycle, we aim to emphasize the lack of "out-of-the-box" direct application.

With this in mind, it is necessary to catalogue and understand better how co-simulation can be used and, perhaps more often, when it is reliable. The example highlights that a model created with the same tool and language will yield different results, what happens when languages and tools become more heterogeneous? Some works address this issue and aim to solve issues with the standard [2,5], in addition to the standard evolving itself. However, much of the literature is, at this point, a decade old, and still, many of the same issues remain with the standard observed at that stage. Notably, the worsened performance from utilizing FMI (particularly version 2) is significant, also including increased execution time (from seconds to hours in cases). For many purposes, the quality of execution with FMI is unacceptable. It cannot be used, even considering the early stages of development, where precise simulation is impractical due to the system's high uncertainty. In this regard, there is a need to understand further when the results of FMI simulation can be considered valid. The standard itself is also evolving with the new third version. Companies utilizing co-simulation today might utilize more extensive libraries of older versions of the standard, and there is a need to evaluate how these can be migrated to a newer version or utilized in joint simulations.

While the notion of abstraction from FMI can enable a broader audience to interact with the models, it simultaneously hides information about the underlying implementation of the models. In addition, while it is possible to reason about the validity of an FMU in a particular domain, it becomes increasingly complex in unfamiliar domains or when models are provided externally. As such, engineers must apply additional knowledge to FMUs to annotate necessary details for re-use purposes. Another issue is the expected increase of shareability across organisations due to the inaccessibility of the underlying models, which should hide implementation details and protect Intellectual Property (IP). However, trust for models which cannot be directly accessed can be difficult to achieve, particularly across teams or organisations. The notion of black box representations is at least a step towards more collaboration due to the ease of hiding sensitive information. However, there are cases of IP leakage regarding FMI, and using models makes it possible to infer information even if hidden behind black box representations [10].

Highlight and summary

The FMI technology has several technical issues that hinder the broader audience from using it off the shelf. The community must classify simulation validity criteria and guidelines to utilise the technology for meaningful simulation and model re-use in development, perhaps via examining FMU extension mechanisms for re-use and classification purposes. Furthermore, there are several concerns with IP leakage in the standard.

4.3 Automation

Referring back to Figure 1, this approach's value is due primarily to the amount of automation included in the transitions. Indeed, the value of automation extends to the overall notion of MBSE. A large part of the benefits compared to traditional SE is the *reduction* of manual activities. In this case, activities do not necessarily refer to the implementation of artefacts but rather the sharing of data across teams. A system and verification engineer might need to invest significant effort to transfer knowledge between models defined for different purposes and abstractions. Due to the FMI standard leveraging abstraction, moving simulation models to a similar abstraction as typical SysML models paves the way for automation.

Leveraging increased simulation model abstraction should improve the integration capabilities between teams and tools, and abstraction, in particular, is an enabler for many MBSE techniques [23]. The addition of simulation can help design and preferably automatically propagate simulation results back to a system model to smooth the process of V&V activities. Indeed, the closely related paradigm of DevOps or other continuous practices like Continuous Integration (CI) or Continuous Delivery (CD) would be a powerful enabling technique [11]. Figure 5 highlights a potential conceptual pipeline loosely based on previous work [4]. In particular, we consider a DevOps process but limit the scope to Design and Evaluation. In this case, the Ops considers model evaluation activities instead of real-world deployment and operation. The Design considers the activities of `Plan`, `Model Creation/refinement`, and `Model Validation`. In essence, the Design considers the planning of new modelling efforts until the eventual validation of the model, which depends on the properties of interest and particular case. This model is primarily SysML based. Evaluation, on the other hand, considers `Model mapping`, `Model Execution`, and `Analysis`. In this stage, we consider the integration of different models (leveraged by the FMI standard), along with the execution and eventual analysis, which propagates back to planning the next iteration, closing the loop. In particular, analysis results can be presented to an engineer through data visualisation in tools (for example, seen in Figure 6) to make design choices in the SysML model.

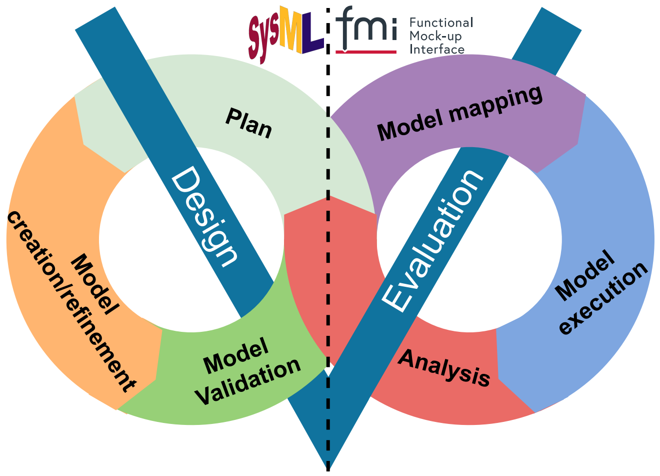

Figure 5: A conceptual DevOps integration for design.

Through this flow, the Design is expected to consider individual modelling efforts in less formal notations. In our case, some or all parts of the `Design` consider SysML diagrams, while the `Verification` considers the automated simulation orchestration and execution of more detailed simulation models. As part of ongoing collaborations, the authors have worked extensively with a CI/CD pipeline for FMU integration[9], aiming to achieve an automated delivery of FMUs. As a case study, a Thermal Management System (TMS) of an electrical machine battery has been the target. We highlight the case in Figure 6.

The original models are created primarily in Simulink and exported as a single FMU. Several different variants exist for the system, which mainly affects the coolant configuration in the TMS. Particularly, the parameters affect one another, and different variants exist with corresponding relationship graphs for the properties. Using typical drive-cycles for Construction Equipment (CE) machines, the aim is to find a suitable configuration to manage the temperature in a specific range of temperatures while aiming to not over-dimension the solution. Apart from the variability concerns, the model considers some starting conditions, for example, ambient temperature, and outputs the battery temperature for each time step. The graph shows three execution traces or different variant configurations over a standard drive cycle of a CE machine. A CI/CD pipeline was created to evaluate the viability of different variant designs and present the result via execution traces to the user using the above-described model. In the example, it is visible that one of the variants produces a very high temperature

[9]See challenge 7: mdu.se/aidoart/events/second-hackathon

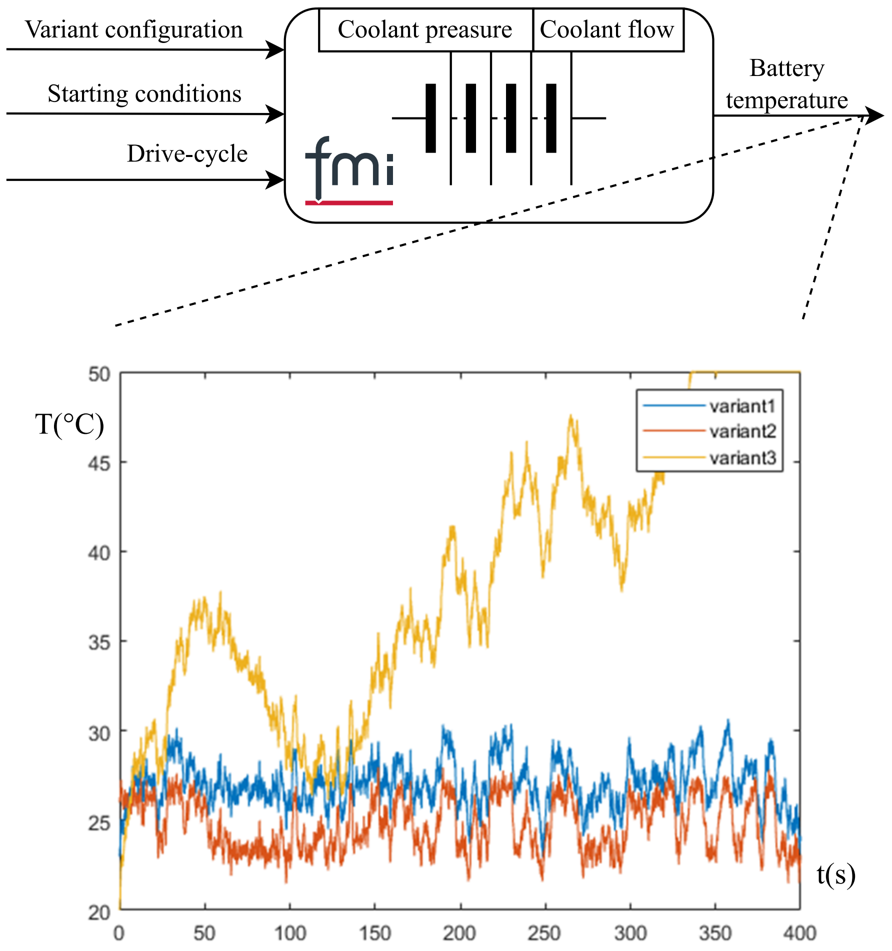

Figure 6: Battery Thermal management case.

while the other two are more similar with lower temperatures. Based on the execution traces, the user (or an automated tool integration) can change the original information source (e.g., a SysML model) by parameterizing the model based on the analysis and possibly instantiating new simulations.

In the solution, simulation models and simulation setup are automatically packed into a "Docker" container[10]. This way, the simulation can run anywhere repeatedly and consistently. The user only needs to select input data to return a different simulation result. Using Docker containers better fits typical DevOps practices and enables complete isolation and portability that FMUs alone still do not fully provide. In fact, FMUs require run-time dependencies to be handled

[10]https://docs.docker.com/

manually, which is against the idea of a simple, standardized model exchange. For instance, let us take a commercial tool as an example of FMU flow usage: building an FMU in MATLAB on a Linux OS will return an FMU with only Linux64 binaries. To run the FMU as-is on a Virtual Machine or a distributed environment will require either a compatible MATLAB version installed or packaging it inside a docker container together with all necessary MATLAB binaries (this is experimentally possible in MATLAB using compiler.package.docker, but the result is a container on a scale of 1 GB for a simple FMU). Both solutions strongly depend on the simulation tool, require maintenance efforts for upgrades, and can create security issues (i.e., having to deploy a relatively old MATLAB image with known security vulnerabilities). The current standard is a difficult fit for an industrial workflow that enables the automation of simulations and data collection using state-of-the-art DevOps tools. Solutions towards this barrier have been proposed in [22], with the release of a command line tool that facilitates the implementation of FMUs in other popular languages that would otherwise not be able to produce C-compatible binaries, associated with a command to configure the generated model to deploy and execute code inside a Docker container.

Furthermore, as FMI evolves, more information is being added to the models themselves, resulting from the difficulty in simulation orchestration with the previous versions. As such, the evolution of FMI is seemingly going *away* from the original black box view to increase the standard's functionality. In this way, enabling meaningful simulation without risking models leaking IP becomes essential. In particular, applying DevOps principles could enable a more sophisticated method of test integration and feedback for the processes in design. Nevertheless, the intersection area between FMU and DevOps tools is relatively unexplored and needs further work.

Highlight and summary

Integrating automation for simulation feedback and orchestration is a vital part of the expected value of FMI, and the DevOps paradigm is a promising outlook. However, the current integration status of DevOps and continuous practices is far from mature and requires more work to realize mature and efficient automated solutions. Notably, there are technical barriers to cross-platform integration that inhibit industrial adoption.

4.4 Value demonstration

When discussing the integration of co-simulation into SE workflows, it is expected to bring *value* to existing processes. Intuitively it seems straightforward that enabling simulation earlier and improving re-use, particularly across domains, is valuable as it can improve V&V, often reflected in literature and industry. Figure 7 highlights a generic example of how knowledge of a system will increase as development progresses and illustrates the potential effect of co-

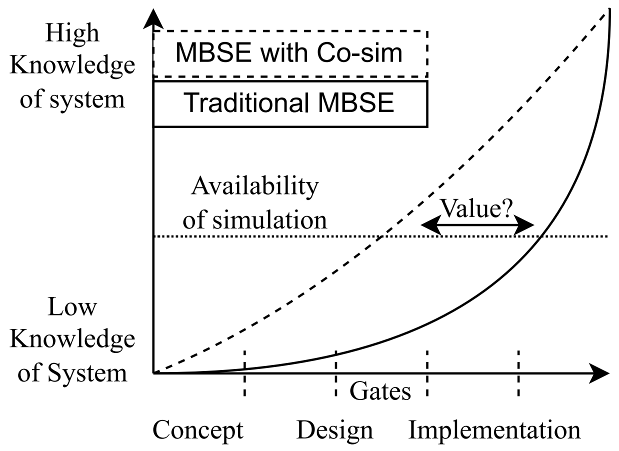

Figure 7: Example of how knowledge increases as the system traverses a (simplified) developmental life-cycle and the potential value of simulation.

simulation and the eventual implementation value. While not perfectly capturing the developmental progress, the main takeaway is that the motivated value comes from reaching specific knowledge or confidence of a system *earlier* than traditional methods. We exemplify this knowledge through the availability of simulation. A large portion of the expected value is that using more automated tooling for re-use and analysis could provide sufficient input for decision-making at earlier stages of development and make existing processes more efficient.

Value is added to the process by introducing more knowledge earlier in development, such as the availability of simulation models. However, the value should be tangible. Value needs to be measured and compared to current best practices and baselines. In the broader context of SE, a current research limitation is the lack of empirical measurements and *evidence* of added value attributed to using MBSE [8, 17]. Similarly, the use of co-simulation mainly reports on benefits which, in the end, are expected and not measured directly, hindering reproducibility and viability in a broader context. Of course, providing tangible measurements of processes and improvements in complex industrial settings that often take place on a time scale of years is challenging. Since much of SE research is applied in industrial settings, it can often be difficult to isolate and truly measure specific aspects of scientific studies, mainly if case studies are interconnected with the studied company. In this regard, the issue of providing empirical value is not a isolated problem for co-simulation with SysML and FMI. Regardless, the community needs to understand the metrics of essence and how they should be measured and reported. For instance, it will often be argued that re-use and interoperability are improved, but that should be made tangible; currently, such aspects are not. As seen previously, this type of solution introduces new issues. Similarly, when discussing SE, the notion of time to market is essential in addition to quality aspects. Added capabilities should be positioned with such concerns to make the connection more concrete with the discipline and overall stakeholders.

From experience with SysML and FMI, we observe that the technology is situated between development areas, such as system architecture and system verification. In this case, the value can often be more easily seen from one side than the other. For system architects, the motivation is relatively straightforward, and SysML maps well to black-box simulation models. For system verification engineers or simulation experts, the value is often less clear when introducing languages like SysML into their workflow. Overall it might, in the end, be a valuable effort from an organisational standpoint. However, technologies that bridge gaps between disciplines require mutual effort, and there is a perceived reluctance from simulation experts to move higher in abstraction in the typical SE flow as they might not see the impact on their processes. A related topic is the cost of supporting co-simulation via FMI. As the models are viewed as black-box, there must be high confidence in the model's validity. Also, there are costs associated with managing FMI-supported model libraries, and suitable models for a given context must be selected for each case. Additionally, tooling support is expected for the SysML to FMI bridge, preferably via seamless automated integration. Such infrastructure is not trivial and will likely add considerable costs in larger industrial contexts. It should be clear what is expected of such surrounding infrastructure and how that factors into the overall value proposition of the technologies, which is often omitted when considering MBSE topics at large.

Henderson *et al.* provide an initial work towards metrics for MBSE and the more comprehensive Digital Engineering (DE) landscapes [16]. In particular, based on previous work, the authors argue that MBSE lacks empirical metrics and extracts preliminary metrics from a systematic literature review. The preliminary metrics are further analysed with MBSE and DE experts to understand the more important categories. The authors conclude by identifying metrics for each of the five success areas of DE, Quality (System quality, Defects), User Experience (System understanding, Effort), Velocity/agility (Time, Rework, Ease of making changes), Knowledge Transfer (Accessibility of information, Collaboration), and Adoption (Project methods/processes, Use of DE/MBSE tools). At a glance, the areas and corresponding metrics seem to apply to the paper's context. Future work could, as such, aim to move towards implementing metrics and pave the way for a more systematic approach to MBSE.

Highlight and summary

The value of introducing the co-simulation capability in MBSE flows is not explicit and hinders rigorous examination while introducing reluctance for adoption. A necessary step for reasoning in this context is the definition of key metrics and costs associated with the technology and the consequent analysis of the implementation effects.

5 Discussion

The motivation and subsequent implementation of the FMI standard in the industry are seen as a valuable effort for the SE community and promote the broader adoption of simulation capabilities. This paper highlights that while the technology is seeing extensive adoption and excitement, it still requires significant work to become a robust part of SE adoption. The standard is seeing significant changes with the FMI 3 version, which promises to solve some identified problems while optimising it overall. Future developments are bound to have similar positive effects, and overall the FMI standard is a step in the right direction to increase interoperability and re-use of models. Many industrial players actively support the standard, and the tool agnostic coupling of models *is* valuable even with the current barriers observed. It is also important to highlight that we situate these barriers in the context of SE, meaning that our observations and experience depend on this assumption. Many of the success stories of FMI come from more simulation-oriented contexts, where most SE concerns are absent or significantly reduced. Furthermore, we have assumed that using MBSE implicitly corresponds to SysML models. While this is a sound assumption, the new SysML V2 standard[11] is bound to impact how MBSE is implemented. The new standard has yet to be widely adopted by industrial tools, which limits industrial usage. However, once SysML V2 becomes a more supported standard, it could significantly impact how models are represented due to the difference in the underlying kernel and largely textually-based language.

This paper generally presents issues that stem from a clash of abstraction, workflow, and engineering responsibilities. The observed context requires engineering roles to ”meet in the middle” in separate processes to boost developmental capabilities. An often necessary condition in this regard is consistency management between the corresponding models at the different abstraction levels. Without maintaining consistency between the artefacts, the activities quickly run the risk of creating inconsistent models, particularly as the level of abstraction is different. Indeed, consistency management between sources of information with vastly different detail and scope is a challenge in itself, reported in literature [19]. In addition to consistency issues, there are challenges related to providing *adequate* simulation models for the early stage. To correctly leverage simulation for analytical purposes, there is a fine line to thread with the use of abstraction to find a suitable trade-off between model detail and the eventual results of the simulation. The issue is tightly coupled to uncertainty and how it should be addressed effectively [7]. Indeed, one could argue that this is the central issue to be addressed, how abstraction should be utilised to enable simulation while preserving meaningful simulation capabilities. The inherent uncertainty requires understanding what can reasonably be extracted from the simulation created at these stages, regardless of the validity of employed simulation models.

[11]https://www.omgsysml.org/SysML-2.htm

6 Conclusion

This paper has described the context and motivation for integrating Co-simulation in MBSE processes. Value is expected through increased collaboration opportunities and a move towards a more mature automated digital thread. In particular, simulation is a capability that can provide more knowledge of a system at earlier stages of development, and the FMI standard is an enabler due to the abstract black box representation of simulation models in an interoperable format. However, several practical barriers are inhibiting the adoption of standards like FMI, particularly from an industrial perspective, presented in the paper. The barriers are summarized into four high-level categories of concerns, *i) Leveraging FMI*, *ii) Automation*, *iii) Integration in SE workflows*, and *iv) Value demonstration*. We exemplify the barrier from industrial experience working in the paper context for each category. Furthermore, we discuss each topic, provide a summary, and highlight to promote future research to reach a more mature technological readiness for industrial use.

Future work could be more organized feedback from the industry in concrete terms towards the discussed issues to enable joint industry-academia collaboration, not only on the technical level of the FMI standard but also in the large SE landscape. A typical pattern for the gaps identified is the lack of rigorous evaluation and a missing unified view of the topics. The authors are working on industrial projects that integrate FMI for analytical purposes and aim to work towards, at least partially, the challenges presented in this paper from the MBSE perspective.

References

[1] K. Abdo, J. Broehan, and F. Thielecke, "A seamless and end-to-end approach for early and continuous validation of next-generation avionics platforms," in *Software Engineering 2023 Workshops*, 2023.

[2] M. Arnold, C. Clauß, and T. Schierz, "Error analysis and error estimates for co-simulation in fmi for model exchange and co-simulation v2. 0," in *Progress in Differential-Algebraic Equations: Deskriptor 2013*. Springer, 2014, pp. 107–125.

[3] M. Ben Ayed, A. Massaoudi, S. A. Alshaya, and M. Abid, "System-level co-simulation for embedded systems," *AIP Adv.*, vol. 10, no. 3, 2020.

[4] J. Bergelin and A. Cicchetti, "Towards continuous modelling to enable devops: a preliminary study with practitioners," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022, pp. 774–783.

[5] C. Bertsch, E. Ahle, and U. Schulmeister, "The functional mockup interface-seen from an industrial perspective," in *Proceedings of the 10th*

International Modelica Conference; March 10-12; 2014; Lund; Sweden, no. 096. Linköping University Electronic Press, 2014, pp. 27–33.

[6] D. Bilic, D. Sundmark, W. Afzal, P. Wallin, A. Causevic, C. Amlinger, and D. Barkah, "Towards a model-driven product line engineering process: An industrial case study," in *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference*, 2020, pp. 1–11.

[7] L. Burgueño, P. Munoz, R. Clarisó, J. Cabot, S. Gérard, and A. Vallecillo, "Dealing with belief uncertainty in domain models," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–34, 2023.

[8] K. X. Campo, T. Teper, C. E. Eaton, A. M. Shipman, and M. B. Bhatia, Garima, "Model-based systems engineering: Evaluating perceived value, metrics, and evidence through literature," *Systems Engineering*, 2022.

[9] P. De Saqui-Sannes, R. A. Vingerhoeds, C. Garion, and X. Thirioux, "A taxonomy of mbse approaches by languages, tools and methods," *IEEE Access*, 2022.

[10] E. Durling, E. Palmkvist, and M. Henningsson, "Fmi and ip protection of models: A survey of use cases and support in the standard." in *Modelica*, 2017, pp. 132–036.

[11] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *Ieee Software*, vol. 33, no. 3, pp. 94–100, 2016.

[12] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language.* Morgan Kaufmann, 2014.

[13] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: State of the art," *arXiv preprint arXiv:1702.00686*, 2017.

[14] J. Gregory, L. Berthoud, T. Tryfonas, A. Rossignol, and L. Faure, "The long and winding road: Mbse adoption for functional avionics of spacecraft," *Journal of Systems and Software*, vol. 160, p. 110453, 2020.

[15] R. Hällqvist, R. Braun, and P. Krus, "Early insights on fmi-based co-simulation of aircraft vehicle systems," in *15:th Scandinavian International Conference on Fluid Power, June 7-9, 2017, Linköping, Sweden*, vol. 144. Linköping University Electronic Press, 2017, pp. 262–270.

[16] K. Henderson, T. McDermott, E. Van Aken, and A. Salado, "Towards developing metrics to evaluate digital engineering," *Systems Engineering*, vol. 26, no. 1, pp. 3–31, 2023.

[17] K. Henderson and A. Salado, "Value and benefits of model-based systems engineering (mbse): Evidence from the literature," *Systems Engineering*, vol. 24, no. 1, pp. 51–66, 2021.

[18] Y. Hirano, J. Ichihara, H. Saito, Y. Ogata, T. Sekisue, and S. Koike, "Toward the actual model exchange using fmi in practical use cases in japanese automotive industry," 2019.

[19] R. Jongeling, F. Ciccozzi, J. Carlson, and A. Cicchetti, "Consistency management in industrial continuous model-based development settings: a reality check," *Software and Systems Modeling*, vol. 21, no. 4, pp. 1511–1530, 2022.

[20] A. Junghanns, C. Gomes, C. Schulze, K. Schuch, R. Pierre, M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson, T. Sommer *et al.*, "The functional mock-up interface 3.0-new features enabling new applications," in *Modelica Conferences*, 2021, pp. 17–26.

[21] E. A. Lee and M. Sirjani, "What good are models?" in *International Conference on Formal Aspects of Component Software.* Springer, 2018, pp. 3–31.

[22] C. M. Legaard, D. Tola, T. Schranz, H. D. Macedo, and P. G. Larsen, "A universal mechanism for implementing functional mock-up units," in *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, ser. SIMULTECH 2021, 2021.

[23] A. M. Madni and M. Sievers, "Model-based systems engineering: Motivation, current status, and research opportunities," *Systems Engineering*, vol. 21, no. 3, pp. 172–190, 2018.

[24] T. Mens, "On the complexity of software systems," *Computer*, vol. 45, no. 08, pp. 79–81, 2012.

[25] R. Mikelöv and A. Bergsten, "Evaluating the reliability of fmi co-simulation for validation," 2023.

[26] A. Morkevicius, A. Aleksandraviciene, and Z. Strolia, "System verification and validation approach using the magicgrid framework," *INSIGHT*, vol. 26, no. 1, pp. 51–59, 2023.

[27] C. Nigischer, S. Bougain, R. Riegler, H. P. Stanek, and M. Grafinger, "Multi-domain simulation utilizing sysml: state of the art and future perspectives," *Procedia CIRP*, vol. 100, pp. 319–324, 2021.

[28] M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, "Simulating sysml models: Overview and challenges," in *10th System of Systems Engineering Conference.* IEEE, 2015, pp. 328–333.

[29] G. Schweiger, C. Gomes, G. Engel, I. Hafner, J.-P. Schoeggl, A. Posch, and T. Nouidui, "Functional mock-up interface: An empirical survey identifies research challenges and current barriers," in *Linköping electronic conference proceedings*, vol. 154, 2018, p. 15.

[30] J. Shi, J. Wan, H. Yan, and H. Suo, "A survey of cyber-physical systems," in *2011 international conference on wireless communications and signal processing (WCSP)*. IEEE, 2011, pp. 1–6.

[31] V. Singh and K. E. Willcox, "Engineering design with digital thread," *AIAA Journal*, vol. 56, no. 11, pp. 4515–4528, 2018.

[32] J. Suryadevara and S. Tiwari, "Adopting mbse in construction equipment industry: An experience report," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 512–521.

[33] A. Vogelsang, T. Amorim, F. Pudlitz, P. Gersing, and J. Philipps, "Should i stay or should i go? on forces that drive and prevent mbse adoption in the embedded systems industry," in *Product-Focused Software Process Improvement: 18th International Conference, Innsbruck, Austria, November 29–December 1, 2017*. Springer, 2017, pp. 182–198.

[34] D. D. Walden *et al.*, "Systems engineering handbook: A guide for system life cycle processes and activities," 2015.

[35] A. W. Wymore, *Model-based systems engineering*. CRC press, 2018, vol. 3.