JOURNAL OF AEROSPACE INFORMATION SYSTEMS

[KPR+25] H. Kausch, M. Pfeiffer, D. Raco, B. Rumpe, A. Schweiger: Enhancing System Model Quality: Evaluation of the Systems Modeling Language (SysML)-Driven Approach in Avionics. In: Journal of Aerospace Information Systems, Volume 22(2), pp. 1-12, American Institute of Aeronautics and Astronautics (AIAA), Reston, VA, USA, Feb. 2025.

Enhancing System Model Quality: Evaluation of the Systems Modeling Language (SysML)-Driven Approach in Avionics

Hendrik Kausch,* Mathias Pfeiffer,* Deni Raco,† and Bernhard Rumpe‡
RWTH Aachen University, Aachen, 52074 North Rhine-Westphalia, Germany
and
Andreas Schweiger§
Airbus Defence and Space GmbH, Manching, 85077 Bavaria, Germany

https://doi.org/10.2514/1.I011476

Quality of software strongly depends on the quality of the models that are used throughout the phases of a development process. Therefore, ensuring model quality is an important aspect that makes a significant contribution to product quality. To determine its contribution to product quality, model quality must be evaluated and demonstrated using suitable indicators. In this paper, the influence of a holistic systems development approach on model quality is evaluated using a use case from the avionics domain. The model artifacts developed in Systems Modeling Language (SysML) version 2 are evaluated in terms of internal, external, and notation quality. A total of 26 indicators are considered to determine model quality. Whereas internal indicators are defined for individual model artifacts, external indicators are defined over multiple model artifacts. Several of the indicators are positively influenced by the expressiveness of the modeling language and the underlying mathematical semantics. Other indicators depend on the methodological guidelines that control the development process. In summary, it can be stated that the application of the presented development approach contributes to high model quality and thus high product quality.

I. Introduction

A. Models in Computer Science

THE size [1,2] and the degree of complexity of software are continuously increasing. These trends make it more and more difficult for software development engineers to fully understand a software system with the aforementioned properties in order to build, update, or maintain it. To achieve the needed manageability of a complex software system (i.e., systems composed of numerous interacting components, hierarchical layers, huge state-spaces, and large input sets) adequate mechanisms (e.g., divide and conquer or suitable modeling) are necessary. Modeling is one of the essential tasks of computer science [3–5]: Real-world elements (e.g., messages) need to be mapped to an adequate model for rendering it processable by a computer. To this end, a model is expressed by a particular syntactic representation (e.g., programming language). The representation abstracts from the object under consideration to depict the properties relevant for the intended purpose. Accordingly, the resulting models can be characterized with the following *properties* [6]:

1) *Image:* Models are always related to an original that they represent.

2) *Abbreviation:* Models generally do not capture all attributes of the original they represent, but abstract from them as required for the respective purpose or use case.

3) *Pragmatism:* Models are not uniquely assigned to their originals per se, but fulfill a replacement function for certain subjects under certain restrictions (purpose and boundary conditions of the model construction). This means that models are subject to variability with regard to the aspects mentioned in this bulleted list.

In addition to this differentiation, a distinction can be made between descriptive and prescriptive models, each of which describe their *purpose*:

1) A *descriptive* model describes an original for easier understanding of a fact (e.g., a city map).

2) A *prescriptive* model describes the way in which an original is going to be created (e.g., a development plan).

In practice, the *graphical representation* of models is given a significant value, although it can be regarded as syntactic shorthand in models (i.e., syntactic sugar) only, because the corresponding semantics is ideally given by the underlying mathematical foundation and the visualization is only another form of representation of the same properties. Nevertheless, the importance of graphical representation is underpinned by the broad acceptance of semiformal modeling languages such as Unified Modeling Language (UML) for software [7] or Systems Modeling Language (SysML) for system development in both the academic and industrial domains. Software development in safety-critical domains is guided by standards such as ED-12C [8]. The current version SysML v2[¶] [9] is in the beta phase. In addition to the expansion and updating of the graphical syntax, the main new features compared to the previous version are a textual representation and more precise semantics.

A model is the most important artifact of model-driven development, so its quality is crucial to the success of a project [10]. For this reason, quality assurance guidelines such as the IEC 61508 [11] standard rightly propose an adequate quality of models in the development process. In particular, quality requirements for models also include requirements for the modeling notation in use.

B. Contribution of the Paper

The objective of this paper is to evaluate whether a model-driven approach can increase model quality in the avionics domain. Model-driven development is an indispensable means of mastering the increasing complexity of software. Because product quality is determined by the quality of the models, guidance must be provided on how this quality can be achieved. In this way, model-driven development can generate positive effects on product quality. Furthermore, because some system properties cannot be fully verified by tests and can only be

Received 17 May 2024; accepted for publication 29 October 2024; published online Open Access 23 January 2025.

*Research Fellow, Chair of Software Engineering.
†Research Fellow, Chair of Software Engineering; raco@se-rwth.de (Corresponding Author).
‡Professor, Chair of Software Engineering.
§System Architect, Embedded Real-Time Software Development.

¶SysML v2 Official Specification https://github.com/Systems-Modeling.

verified by using formal methods, the model-driven approach should also cover the integration of formal methods. A model-driven approach is defined as the (Modeling-Language, Methodology, Tool).

C. Structure of the Paper

Suitable model-quality attributes are determined from literature and presented in Sec. II. Our presented approach and its related work will be detailed in Sec. III. It provides the necessary semantic foundation, offers a tool, includes a development methodology to mitigate the risks of formal verification, and increases the success rate of such a verification. Our conducted study is presented in Sec. IV. The results in Sec. V show the extent to which the identified model-quality properties can be addressed. It is examined whether a model-driven development approach can generate a positive influence on model quality. Section VI then discusses the findings and particularly focuses on indicators not found to be improved or even found to be diminished. The results are summarized and an outlook for future work is provided in Sec. VII.

II. Model Quality Attributes

In Table 1, quality characteristics for models are summarized and explained according to the taxonomy of Fieber/Huhn/Rumpe [10]. In a European industry collaboration [12], this taxonomy has already been used to define a process that ensures the quality of software models. This combines various metrics, unusual constructs in a model that indicate potential problems, inefficiencies, or issues with its design, structure, or usage (i.e., smells), and refactorings. Even tool support is taken into account. In [13], this approach is pursued further and, among other things, a tool environment for automated analysis is provided. In contrast to this automated approach, the authors of [14] take the path of manual reviews, but continue to refer to the taxonomy and approaches of the previous work. Ref. [15] also takes up the taxonomy and uses it to adequately create models for road safety. As an extension of [16], this taxonomy therefore forms the basis for the evaluation (see Sec. V) of the approach for model-driven development and formal verification presented in Sec. III. The quality attributes described in Table 1 can be applied to any model-driven development approach, for example, the approach introduced in Sec. III. The quality attributes are a compilation and extension from [10]. All quality attributes presented in [10] were included in Table 1.

Most quality attributes are separated into model-quality properties that can be defined individually for each model (intramodel properties) and cross-model-quality properties concerning multiple models (intermodel properties). Intermodel properties are further refined into model-quality properties applying to models of only one granularity level (horizontal), or to models over multiple granularity levels (vertical). Granularity levels are created in the iterative system development process as soon as the system is described at a different (e.g., deeper) level of abstraction. Granularity levels of the system modeled later have less underspecification or are further decomposed. Recurring model-quality properties, for example, consistency, can be relevant for an intramodel scope as well as for an intermodel scope. Whereas intramodel consistency might be checked by ensuring type-correct usage of variables, intermodel consistency, for example, that the interface of two related models match, is a different kind of consistency. Such recurring quality properties in Table 1 are evaluated separately.

III. Model-Driven Approach

The model-driven approach evaluated in this paper is the triple (MontiBelleML, MontiBelle, F-IDE [Formal Integrated Development Environment]). The *methodology* MontiBelle is a development process derived from SPES [19–21]. This provides a framework for model-driven and formal verification. Its basis is formed by FOCUS [22,23] as a semantic foundation for the interpretation of system models. It also includes an automated tool F-IDE using MontiBelleML (subset of textual SysML v2[¶] models) as an input.

A. Semantic Foundation

In the formal specification language FOCUS, distributed and interactive systems consist of components that exchange messages with each other via directed channels. The semantics of a component is defined as a set of stream-processing functions, where each function of the set represents a potential behavior of the component. The refinement of the behavior is represented by the subset relation ($\subseteq$). Concurrency or parallelism of the system is covered by a composition operator ($\otimes$) that connects channels. The most important reason for using FOCUS as the basis for MontiBelle is the property that the refinement is fully compositional [23,24]. This means that after decomposing a system, its components can be refined separately and then reassembled. By design, the composite system is a correct refinement of the system before the refinement of its subsystems. This feature significantly reduces the testing and integration effort. Figure 1 shows a FOCUS architecture of the alternating bit communication protocol. Messages of the `Data` type are transmitted from the sender input `i` to the receiver output `o`. Internally, ds and dr messages are tuples of *Data* and `Bit`. The receiver returns an acknowledgement bit *ar* and *as* over a *Medium* to the sender. Refinement of a subcomponent, for example, the Medium component, automatically leads to the refinement of the whole system.

B. Methodology

The methodology MontiBelle is a development process derived from SPES [19–21] and SpesML[**] that can be used in the context of the EUROCAE ED-216 [25] standard, which forms the basis for the use of formal verification in the development of software systems for avionics. The methodology aims at providing users with the necessary guidelines to apply formal verification intuitively and successfully in systems development. The MontiBelle methodology uses model-driven concepts to enable abstraction, where possible. At the same time, formal specification techniques enable fine-grained control over system specifications, where required.

The MontiBelle methodology covers the early phases of development and thus generates the positive effects of front-loading [26]. As such, it provides a means of ensuring conformity, consistency, verifiability, and traceability between system requirements (SRs), high-level requirements (HLRs), software architecture design [27], and low-level requirements (LLRs). First of all, informal SRs are typically formally specified in a declarative way during development via communication histories in the form of HLRs. For this purpose, the developer specifies input–output relations in the assumption-guarantee style.[††] According to EUROCAE ED-216, the further design of a system is divided into two activities: Deriving an architecture a nd developing the LLRs. The MontiBelle methodology proposes to decompose the HLRs into more detailed and specialized architectures until they are fine-grained enough for developing the software.

During such decompositions of declarative specifications into other declarative specifications or architectures [27], refinement proofs to the previous granularity level must be performed to demonstrate that the refinement is correct. The compositionality of the refinement in FOCUS enables the correctness of their composition to be derived automatically from the correctness of individual components.

Finally, architecture and LLRs are combined into a system design. This step is carried out by refining the most granular HLR layer into a structurally identical LLR architecture. The correctness of this refinement must be proven. Thanks to the compositionality of FOCUS, however, this correctness already follows from the individual proofs of the refined components. These proofs are fully automated in the tool.

In summary, we present a methodology levering generative model-based formal verification. It covers the early development phases and provides means to guarantee the compliance, consistency, verifiability, and traceability among SRs, HLRs, and the design of a software architecture, as well as LLRs, as shown in Fig. 2.

C. Tool

The proposed *tool* F-IDE is represented in Fig. 3. Models of a SysML v2 profile called MontiBelleML describe systems at different levels of abstraction. Between the different levels of abstraction,

**SpesML Project Site https://spesml.github.io.

[††]A specification style that guarantees certain properties (guarantee) as long as the associated preconditions (assumptions) are met.

Table 1 Quality properties for models ([10], pp. 416-420), extended with vertical quality attributes, verifiability and transformability, and a brief explanation thereof

Intra-model-quality properties	
Representation	Presentation measures how easily information can be grasped cognitively. Better presentation implies better understanding. An aesthetic and structured presentation makes it easier to grasp a model's information.
Precision	Precision measures the extent to which all relevant properties of the modeled system are represented. Precision thus addresses the reduction or condensation of information as a result of the modeling.
Universality	Universality measures the extent to which the models focus on defining the relevant details. Low universality could result from an unnecessary focus on platform-specific details in early development phases and lead to complex hardware solutions later on.
Simplicity	Simplicity measures the extent to which an issue is not presented in a more complex way than is actually necessary. Simplicity can be increased without any loss of information, e.g., by using reformulation, restructuring, or abstraction.
Semantic adequacy	Semantic adequacy expresses the suitability of a model that appropriately represents the desired information. Entity–relationship models, for example, are well suited to representing entities and their relationships. However, they are less suitable for modeling the behavior of software components.
Consistency	Consistency measures the degree of absence of errors within a single model element. Typical (internal) consistency checks include the declaration of variables before their type-correct assignment.
Conceptual integrity or uniformity	Conceptual integrity or uniformity aims at providing similar solutions to similar problems. This quality indicator measures the degree of similarity of modeled solutions within a single model element. Conceptual integrity can be achieved by applying repeatable rules, patterns, and principles.
Conformity	Conformity measures the degree to which standards and norms are applied to models. Conceptual integrity can thus be a consequence of conformity, provided that the standards and norms set relatively narrow limits. Typically, project-specific guidelines are required to achieve conceptual integrity based on conformity.
Language-specific, semantic quality indicators	Language-specific, semantic quality indicators are specific for the respective modeling language. Examples are the completeness of state transition diagrams or the liveness in Petri nets [17].
Horizontal intermodel-quality properties	
Consistency, conceptual integrity, language-specific, semantic quality indicators	Consistency, conceptual integrity, language-specific, semanticquality indicators are also relevant across models. In addition, the definition and visibility of interfaces are examined here, as well as the management and correct cross-model use of interface elements.
Downward completeness	Downward completeness of models is achieved if they contain all the information required to create the model artifacts of the next development phase in the model chain.
Cohesion	Cohesion expresses the extent to which content-related or technically related parts and facts are represented in closely linked model artifacts.
Modularity	Modularity specifies the extent to which model parts only represent individual aspects. Modularity increases the reuse of model artifacts.
Freedom from redundancy	Redundancy can produce a lot of extra work in an iterative development process. Even if development with complete freedom from redundancy is neither possible nor desired (see controlled redundancy), redundancy between models should still be minimized.
Controlled redundancy	Controlled redundancy allows targeted redundancy, e.g., for modeling different views of system parts or facts [18].
Vertical intermodel-quality properties	
Correctness	Correctness defines the degree to which a model correctly implements the requirements of earlier artifacts in the model chain. This could include that the requirements of earlier development phases are correctly developed against an implementation.
Downward completeness	Downward completeness expresses the extent to which all modeled requirements of a coarser element in the model chain are refined in the current element of the model chain.
Upward completeness	Upward completeness measures the extent to which the requirements of a previous development phase have been fully met. Upward completeness is closely linked to correctness (as a requirement has a significant influence on operational safety), because only correct derivations result in completeness. However, correctness does not induce completeness. For this purpose, requirements of an upstream development stage are considered, which are to be developed into a large number of new models. Correctness can be measured individually for each new model. However, completeness requires consideration of the entire model set.
Traceability	Traceability measures the ability to track information usage from a model, enabling creation of a new model or parts in a subsequent development phase.
Modifiability	Modifiability measures the ability to maintain, extend, or reuse models or model elements (for correction or further development).
Freedom from vertical redundancy	Freedom from vertical redundancy determines the extent in models to which redundancies between successively developed model artifacts are minimized.
Controlled redundancy	Controlled redundancy is defined as targeted or advantageous redundancy, e.g., to model differently granular views of system parts or facts. It allows the semantic consistency of requirements between elements of the model chain of successively developed model artifacts to be checked.
Quality requirements for modeling notation	
Degree of formalization	Degree of formalization measures the use of formal description, whereby the rules of a formal language are adhered to. An increased level of formalization increases the ability to analyze, simulate, or generate artifacts from models. Later development phases generally require a higher degree of formalization.
Adequacy for the application domain	Adequacy for the application domain measures the extent to which a particular modeling notation can be used in modeling typical domain concepts. Models should generally be clear and compact while retaining the ability to model even complex, domain-specific concepts. Inadequate notations typically lead to unnecessarily complex models, or even concepts that cannot be modeled at all.
Further model-quality attributes	
Verifiability	Verifiability determines the degree by which modeled properties can be verified in relation to their correctness. Verifiability requires a certain degree of formalization, because models and their properties require defined semantics for carrying out the verification. A special case of verification is testing, which requires an executable model.
Transformability	Transformability measures the extent to which models can be typically, but not necessarily, processed and transformed by a machine. Transformations can be used to derive simulations or a (formal) analysis.

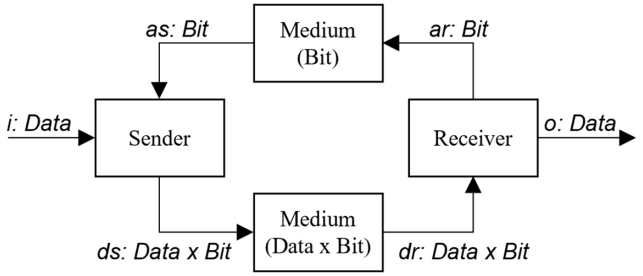

Fig. 1 FOCUS architecture of the alternating bit communication protocol.

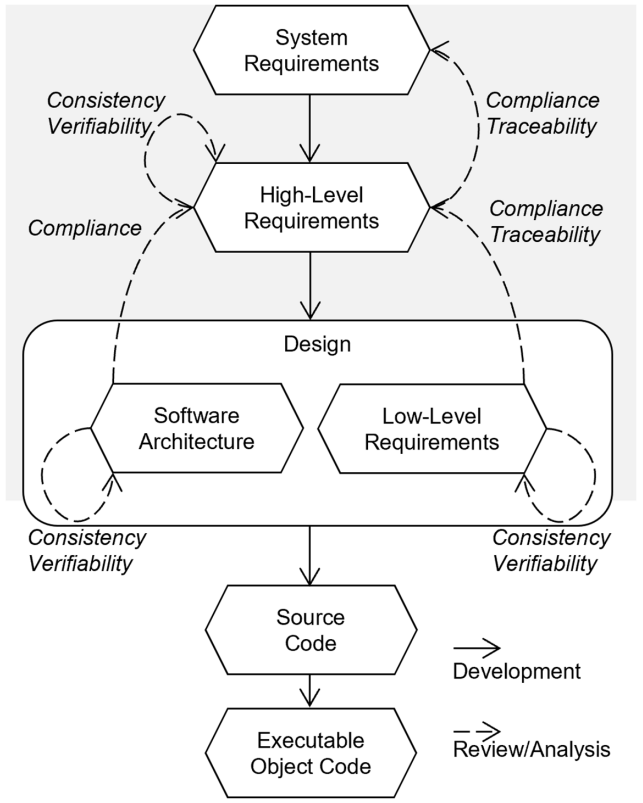

Fig. 2 Systematic design of the MontiBelle methodology (adapted from EUROCAE ED-216).

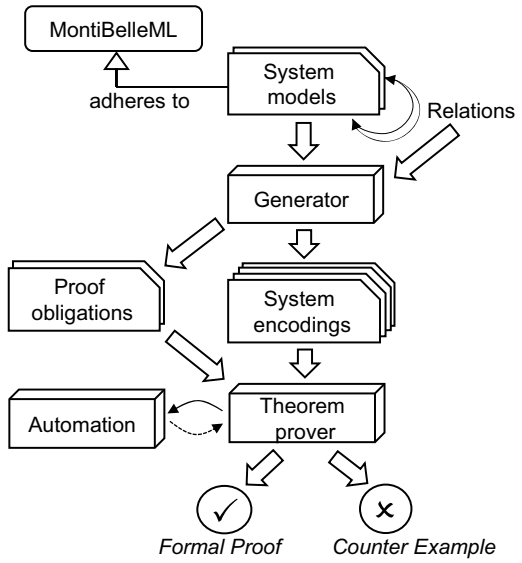

Fig. 3 Overview of the tool.

modeled refinement relationships specify the properties to be checked. A generator translates these models and refinement relationships into the syntax of a theorem prover (Isabelle [28]). Theorem provers such as Isabelle enable the machine-supported and automated search for proofs. In particular, the presented approach uses the generated theories together with the general theories (FOCUS encodings) to perform the formal verification of the refinement relations. The verification can be carried out autonomously with the help of automation scripts. The results are machine-verifiable proofs of correctness or, in the case of errors, comprehensible counterexamples.

D. Related Work

Bansiya and Davis [29] developed an approach for the quantitative evaluation of the design properties of object-oriented designs. They calculate high-level quality attributes from the corresponding metrics. However, the approach only applies to object-oriented systems, whereas model-driven development is the more general approach targeted by this paper.

The model-driven approach was applied to use cases from domains such as the automotive and aerospace industries. Advantages for the practitioners include significant reduction in testing and review effort. In [30], a synchronous time model was used and evaluated using a case study from the automotive industry. The synchronous time model is known to be better suited for the specification of hardware systems. The Isabelle implementation of FOCUS was connected to a frontend of the domain-specific language MontiArc [31].

MontiArc and a synchronous time model were used again in [24] to formally prove the properties of a pilot flying system from a NASA case study [32]. A nondeterministic, underspecified variant of the same case study was gradually refined in [33] and the refinement was verified for correctness. MontiArc and a synchronous time model were used.

In [34], the industry-standard language SysML v2 was used as the frontend, again in combination with a synchronous time model. Later, an event-driven time model was introduced in [35], which is better suited for modeling software systems. The "pilot flying system" was again modeled in this context using SysML v2. A data link case study from the avionics domain was verified in [36] using the event-driven time model. An in-depth view for the data link case study is given in [37]. Additionally, the data link case study was modeled in MontiArc in [38] and the basics of the event-driven time model were explained in more detail in FOCUS. An extension of the model-based approach with an artificial intelligence for creating trustworthy system models was researched in [39].

Similar approaches that use synchronous data-flow modeling languages such as Lustre [40] have been developed for the specification of reactive systems. However, their synchronous time model is more suitable for describing hardware systems. There are also other prominent approaches for modeling distributed systems, for example, the Palladio Component Model [41], AutoFocus [42], MechatronicUML [43], or the Ptolemy Project [44]. Whereas Ptolemy deals with the design and simulation of deterministic and continuous systems, our model-driven approach focuses on discrete and underspecified systems and the mathematical verification of correct behavior. Palladio also uses simulation to analyze distributed systems in terms of performance and timing properties. AutoFocus concentrates on discrete systems, but in contrast to the approach presented here, it only supports a time-slice-driven specification of systems, whereas MontiBelleML also enables event-driven modeling. Similar to the approach presented here, MechatronicUML can be used to model distributed systems using different diagrams and system views in an iterative development process. Formal verification is also possible by connecting model checkers and simulations. An integration of theorem provers does not yet exist in MechatronicUML.

IV. Avionics Case Study

In this case study, a wireless avionics data link [36,38] is developed using the presented model-driven approach. The Data Link Uplink Feed (DLUF) system is used to transmit prioritized data packets via a

connection, for example, between an unmanned aerial vehicle and a ground station. This system is representative of software systems in the avionics domain and is used in this paper to evaluate the model quality resulting from the application of the approach. The developed system shall fulfill the provided SRs. For a *liveness* SR, instead of only demonstrating the correct functionality based on tests, a formal proof is provided as an example. This shows that the required property is valid for the entire system across all scenarios. Complete coverage by tests is not possible here, as this property is formulated over an infinite time period. Furthermore, the property cannot be adequately covered by verifying the individual system parts, but requires the verification of the system property across the entire system, that is, by integrating all system artifacts.

Along the MontiBelle methodology, the liveness property is encoded in a descriptive model using the textual notation of SysML v2. The specification is created in a part definition in SysML notation and is represented in Fig. 4.

The keywords `part def` define such a part definition in the first line. The interface of the system is modeled in Lines 2 and 3 via ports. The keyword `port` defines ports, followed by any name, here `input` or `output`, which transmits elements of a packet data type. The direction of the port is controlled via so-called conjugations (~). Cardinalities in square brackets, here 4, allow ports to be duplicated without considerable effort. The HLR is formulated descriptively from Line 5 in a `requirement` block. This comprises three assumptions, formulated with the keyword `assumes`, and a resulting guarantee, modeled with the keyword `requires`. The assumptions and guarantees are formulated using textual expressions that are strongly based on well-known mathematical expressions. The concrete interpretations correspond to the stream expressions from [23]. The specification is also parameterized using the constants `MAX_CAP` to cover various capacity limits.

Starting from the HLR, fine-grained specifications were developed using decomposition. All the resulting subsystems are also described using descriptive models similar to those in Fig. 4. This methodology of developing HLRs from SRs is compliant with EUROCAE ED-216. The modeling of traceability is enabled by using the keyword *refines*. This is a derivative of the general specialization relation of SysML (specialization). As soon as a sufficiently granular decomposition level is reached, the prescriptive (LLR) models are then defined for each undecomposed (atomic) component. The traceability between prescriptive and descriptive models is also indicated via the relation *refines*. In the case study, the system modeled with the LLRs consists of eight atomic components, four buffer components, and four capacity gates.

```
1  part def DLUF_HLR1 {
2   port input: ~Packets[4];
3   port output: Packets[4];
4
5   satisfy requirement 'non-starvation' {
6    assumes 'infinitely long timeframe' {
7     ∀i∈{1,2,3,4}:
8      input[i].length() = ∞
9    }
10   assumes 'message for each interval' {
11    ∀i∈{1,2,3,4}, t:nat:
12     input[i].atTime(t).length() > 0
13   }
14   assumes 'size below max. capacity' {
15    ∀i∈{1,2,3,4}: ∀v∈input[i].values():
16     v < MAX_CAP[i]
17   }
18   require 'infinitely many outputs' {
19    ∀i∈{1,2,3,4}:
20     output[i].messages().length() = ∞
21   }
22  }
23 }
```

Fig. 4 Listing 1 descriptive liveness HLR, which is formally modeled in the textual notation of SysML v2.

A graphical representation of the LLR buffer components in SysML v2 is depicted in Fig. 5. An automaton is specified with the keyword `state` followed by any name. This is followed, separated by a horizontal line, by the definition of an attribute `b`, which corresponds to the type `List<Packet>`, that is, a list of packages. The keyword `entry` allows the definition of an initial action. Specifically, the internal list `b` is initialized with the empty list. This is followed by a diagram that describes the other states and transitions. The only state `S` is also the start state, marked by an incoming transition without an initial state (black dot). Transitions are represented by arrows from the initial state to the target state. Transitions consist of a trigger, a guard (represented in square brackets), and, separated by a slash, an action. The buffer automaton is triggered by a message event on a channel, recognizable by the channel name as a stimulus, or time progression, represented by the stimulus `Tick`. The guards allow the transitions to be activated only conditionally. The guards are textual, Boolean expressions, comparable to those from Fig. 4. Finally, the reactions allow the assignment of new values to internal attributes using the keyword `assign` or the sending of messages using expressions of the type `send-to`. Similarly, a capacity gate in Fig. 6 is represented as an automaton.

Capacity gates have the two internal states *Send* and *Block*. The current state shows whether there is still enough capacity to send messages in the current time slice. This depends on the capacity attribute *cap*, which describes the current limit value. If this is exceeded by the size of an incoming packet, packets are blocked until the end of the current time slice. In such a case, the control message *Nack* informs the buffer component about the blocking. At the end of a time slice, the capacity is reset to the original value and messages can continue to be transmitted.

All prescriptive models are assembled into an LLR system by means of composition. There are a total of four scheduler subsystems, each consisting of a buffer and a capacity gate. By using parameterization of the maximum capacity as a variable `MAX_CAP`, the specification of the capacity gate can be reused. The scheduler subsystems are also modeled using parameterization. A graphical representation of the scheduler subsystem is depicted in Fig. 7. The keyword `part def` models a part definition in analogy to Fig. 4. This part definition is composed of two subsystems or `parts` whose ports are connected using connectors (arrows). For example, the output port `o` of the buffer is connected to the input `i` of the capacity gate. The methodical decomposition and refinement of the models lead to a tree structure that represents both the development path and the verification artifacts (proof obligations). As part of the case study, all these proof obligations were met in an automated way. Figure 8 visualizes the refinement relations.

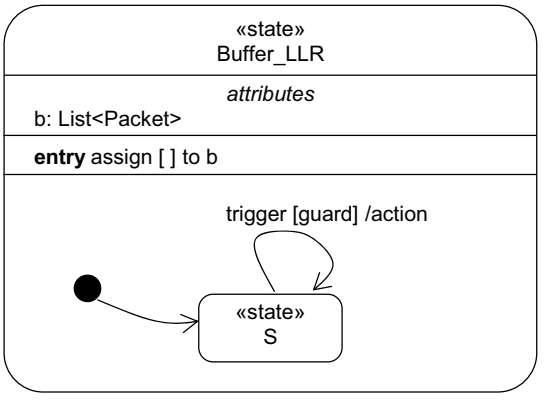

Fig. 5 Graphical representation of the prescriptive buffer model in SysML v2.

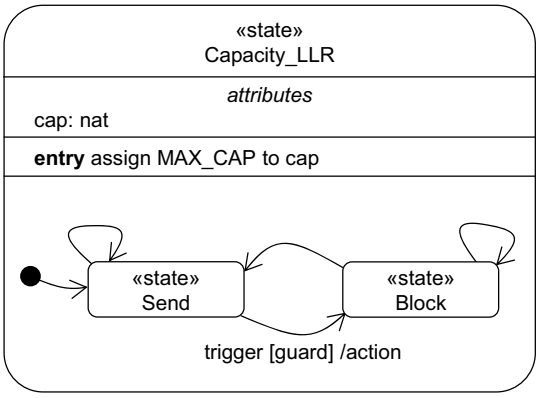

Fig. 6 Graphical representation of the prescriptive capacity gates in SysML v2.

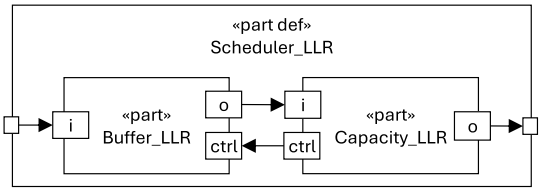

Fig. 7 Graphical representation of a scheduler subsystem consisting of a buffer and a capacity gate in SysML v2.

A complete traceability overview for the DLUF case study, its models, and its refinement relations is given in Fig. 8. The models are grouped into four granularity levels: HLR1, HLR2, HLR3, and LLR. The models of the first level (HLR1) are directly derived and traced to the SR of the case study. The decomposition of the DLUF system into subcomponents is depicted using the dotted arrow and refined by a continuous arrow, for example, the DLUF_HLR2 model composes 3 Schedluer_HLR2 models together and refines DLUF_HLR1.

V. Evaluation

In the following sections, the presented model-driven approach is evaluated from the perspective of the quality properties for models described in Sec. II (see Table 1). This determines to which degree the approach is supportive to which properties.

A. Internal Quality

1. *Representation*

The MontiBelle methodology provides a decomposition and an abstraction concept. Utilizing these concepts, models are less overloaded, as the models focus on parts of the system and can achieve a high degree of abstraction. This is shown in the *Scheduler* specification in Fig. 7. The approach also indirectly increases cognitive perceptiveness by suggesting a modeling language that is widely used in the industry, namely SysML v2, instead of relying on proprietary languages. In particular, the textual syntax of SysML v2 allows for adequate structuring through arbitrary formatting. A graphical representation is also provided.

2. *Precision*

MontiBelleML's stream expressions [36] for requirement specification, together with the decomposition of the system into subsystems and aspects, allow for a concise formulation of properties. In Fig. 4, for example, the first HLR (liveliness) is described in a few lines in an abstract, yet semantically precise manner. Apart from the signature, no other model elements are required to describe the system and its requirements. For implementation-related specifications, MontiBelleML relies on an event-driven, state-oriented specification technique. This is particularly suitable for software-intensive, time-critical systems [45,46]. Due to the decomposition, these specifications are still clear, as the graphical representation of the *buffer* LLR in Fig. 5 shows. At the same time, the model is precise enough to be able to derive an implementation.

3. *Universality*

MontiBelle allows purely history-oriented specification [38] in early development phases. Apart from the signature of the system and the expected inputs and outputs or their relations, no assumptions are made about the implementation of the system. Every implementation that

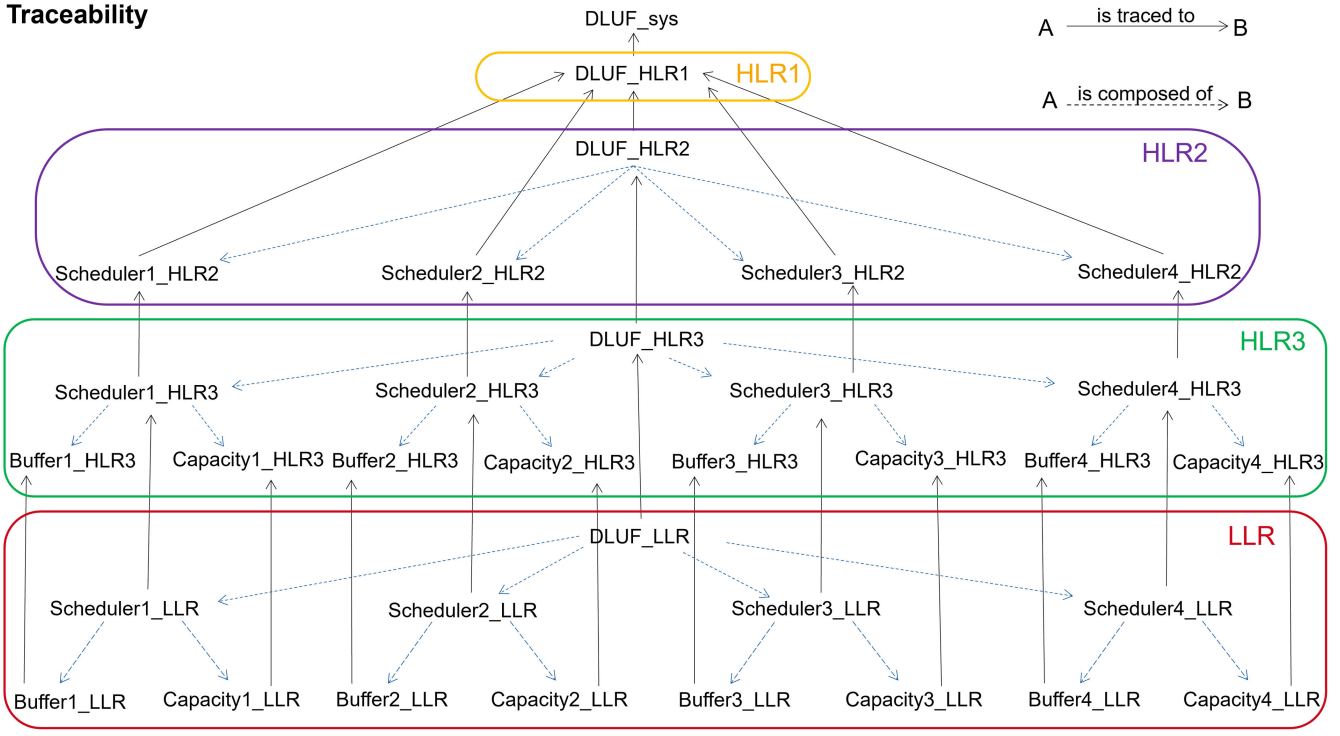

Fig. 8 Hierarchical decomposition of the case study and traceability of the refinement relations.

adheres to this input and output contract is compliant with the system requirements and the system models. This ensures that no platform dependencies are introduced. Even in the later development phases, MontiBelle relies on abstract (platform-independent) state machines. In the case study, for example, a *Buffer* and *Capacity* component were modeled. The graphical representation of the two specifications is depicted in Figs. 5 and 6, respectively. Although platform-dependent code can already be automatically derived from these, suitable generators can support any platform that calculates new outputs based on inputs and the current system state.

Furthermore, MontiBelleML offers the option of leaving models underspecified. This means that several, potentially infinite, valid realizations can exist that fulfill the history or state-oriented contract. The evaluated model-driven approach therefore also supports variability and product line creation and management.

4. Simplicity

MontiBelle suggests simplicity-increasing development of models. It proposes the semantic preservation of a simpler model to be formally verified against a more complex model, thus enabling correct reformulations. These can also represent refinements or abstractions, as MontiBelle is capable of underspecification. One example is the refinement and decomposition of the scheduler component into a simpler buffer and capacity component. This decomposition is depicted in Fig. 7.

In addition, MontiBelle is essentially syntax-agnostic [34,38]. If a domain requires certain assumptions or special forms of representation for complex issues, MontiBelle allows one to achieve greater simplicity through domain-specific adaptations or by switching the modeling language (e.g., by providing a special security keyword to encrypt the transmission or introducing a timing keyword to limit the time delay).

5. Semantical Adequacy

MontiBelle is semantically adequate for modern software systems. Thanks to its meaningful formal verification, MontiBelle is suitable for safety-critical applications and time-critical systems. Software systems exchange data over a potentially long period of time. Time-critical systems require modeling and analysis of time. MontiBelle uses timed communication histories and an event-based processing method that is particularly adapted to software-intensive, time-critical systems (see the explanations on precision in this paper in Sec. V.A, [45,46]).

6. Consistency

Our F-IDE uses MontiCore [47] to process the models and analyze their internal consistency. MontiCore offers the possibility to check models for syntax correctness and supports the development of further analyses. These analyses include, for example, checks for compliance with conventions when naming elements, but also type checks of the (stream) expressions used or existence checks of all states used in transitions. The DLUF models are checked using MontiCore for type-correct use of the interfaces, both when composing distributed systems and when referencing interfaces within a component for behavior specification.

7. Conceptual Integrity/Uniformity

MontiBelleML provides internal conceptual integrity/uniformity through the use of a domain-specific profile for the system modeling language SysML. The profile restricts the use of model elements to those that have a semantic basis in FOCUS. For this purpose, MontiBelleML currently uses history-oriented, descriptive models as in Fig. 4, and state- and event-oriented, prescriptive models as in Fig. 5. However, it is not sufficient to consider only the internal conceptual integrity/uniformity. If the subsystems and views of these systems are considered across a complete granularity level, it is all the more important to achieve this quality feature. The use of the profile is therefore again relevant in Sec. V.B. Nevertheless, the use of the profile already lays the foundation for achieving horizontal conceptual integrity.

8. Conformity

MontiBelle demands compliance with the concepts from Sec. III. Thus, models always consist of the signature of the (sub)system and its behavior, be it descriptive or prescriptive in nature. However, its particular strength lies in the refinement relationship between models and the verifiability of these models. For example, MontiBelle achieves the conformity of the models, which is to be achieved according to EUROCAE ED-216, through external, that is, cross-model, features (see Sec. V.B).

9. Language-Specific, Semantic Quality Requirements

This is the core competence of MontiBelle. Employing a suitable semantic foundation and mapping the models into a theorem prover, the approach achieves the semantic analyzability of the models. This includes the abstract, history-oriented specification style and underspecification.

As already explained in Sec. III, MontiBelle allows for the refinement between system specifications to be verified. Through decomposition and the use of FOCUS, even complex systems can be developed and verified in this way. This verification of the overall requirements increases the quality of the models. This is discussed further in Sec. V.B.

B. External Quality (Horizontal Relationships)

1. Consistency, Conceptual Integrity, Language-Specific, and Semantic Quality Requirements

Some of the aforementioned internal quality attributes can often only be considered from a cross-model perspective. The modeling language developed in MontiCore allows users to describe refinement model dependencies. In DLUF, the necessary refinement relations were added to traceability and their correctness was proven, as Fig. 8 shows. The model interfaces, that is, the interfaces of the modeled subsystems, are well defined and are checked for compatibility with each other. Verified systems therefore have correct, syntactic model relationships, and interfaces of referenced models must be used correctly. MontiBelle promotes external horizontal conformity and conceptual integrity through the FOCUS foundation. The consistency of atomic subcomponents has already been ensured in the context of inner conformity (see Sec. V.A). The model consistency of an entire granularity level follows directly from the consistency of the subcomponents and can therefore be checked automatically.

2. Downward Completeness

With the MontiBelle methodology, a granularity level is downward complete, as soon as the models of the system at this level fulfill all the required properties of the levels above. There are no further requirements, as the approach enables underspecification, that is, any number of details can be omitted to avoid unnecessarily restricting the solution space. For the coarsest model of a system, downward completeness is given according to the MontiBelle methodology, if the correct implementation of the informal requirements has been validated. This is outside the scope of the approach. At subsequent granularity levels, downward completeness is given, as soon as the refinement relation to the previous level is shown. The next granularity level in the model chain can then be developed, that is, finer granular specifications are modeled. For DLUF, the correctness of refinement relations was formally proven across four levels of granularity, as Fig. 8 shows. Due to the underlying mathematical theory FOCUS, the refinement check is also closely related to the upward completeness (see Sec. V.C). To summarize, the approach requires downward completeness as part of its methodology and provides a mechanism to measure this completeness for models.

3. Cohesion

The (de)composition promotes both the cohesion and the modularity of the models. According to the MontiBelle methodology, closely related aspects of the system are treated as individual and focused subsystems, as this is advantageous for successful verification. In the DLUF models, for example, this is the case in the

scheduler elements. There is a strong cohesion between the buffer model (see Fig. 5) and the capacity model (see Fig. 6), as together they specify the scheduler (see Fig. 7). Other models do not need to be considered to understand the scheduler. By methodically applying the language options according to the Montibelle methodology, cohesive parts of the DLUF system are also modeled cohesively.

4. Modularization

(De)composition (see Figs. 5–7) may be even more important for the modularity of the models than for cohesion. The decomposition of a system into several subcomponents creates more and more modularity, as only individual aspects are dealt with in the models of the subcomponents. These subcomponents are then modular enough to be further developed vertically and independently of each other through refinement and without leading to integration problems at the same or higher granularity levels. The reason for this is that, according to FOCUS, the refinement of a subsystem leads directly to the refinement of the overall system. For example, in DLUF, the HLR model of the buffer can be further developed into the LLR model of the buffer without having any influence on the scheduler component or the entire DLUF system.

5. Freedom from Redundancy

A horizontal granularity level of the system models modeled in MontiBelleML leads to redundancy-free models by importing and reusing component models. The scheduler composition in Fig. 7 does not define any buffer or capacity interfaces nor any behavior. Instead, only model artifacts are referenced for the composition. Furthermore, MontiBelleML eliminates additional redundancies by allowing for parametric models. A single-capacity model is sufficient to map the four *different* scheduler components of DLUF [36]. This also further reduces the scope of the necessary verification. However, completely redundancy-free modeling is not possible. For example, a model name is used when referring to other model artifacts. If this model name is changed, the references to this model must also be changed.[‡‡] This means that model names are redundant in several models. Additional tools that track references via model artifacts alleviate this problem.

6. Controlled Redundancy

Model redundancies in the referencing of models enable the static verification of the type and structure correctness of compositions, as the interface types cannot be compared without references to corresponding models. The verification coverage is given, because the approach formally verifies the correctness of a granularity level. Splitting the system into individual models creates associated controlled redundancy and allows these models to be developed independently of each other by different developers. In DLUF, different development engineers could specify and refine the capacity or buffer component models.

C. External Quality (Vertical Relationships)

1. Correctness

For correctness, the requirements of the previous level must have been implemented correctly in the subsequent level. MontiBelle enables the modeling and verification of correctness through the refinement relation and the FOCUS theory. For example, the case study demonstrates that the architecture and the LLRs represent a correct subsequent model level for the previous HLR level. At first, all system specifications of all granularity levels and all refinement relations between them were automatically translated into the syntax of a theorem prover. Then, all proof obligations are fulfilled using automatic solvers and checked automatically [36]. In summary, this proves that the architecture and LLRs are correct in regard to the top-level HLRs. The approach not only verifies this quality property, but also supports the development of correct models. Counterexamples for refinement relations can be found and extracted as test cases, which eases fixing incorrectness in an informed way.

2. Upward Completeness

Orthogonal to correctness, this property requires that all requirements from the previous granularity level have been fulfilled. The argumentation therefore largely follows that of the preceding paragraph on correctness, and the necessary global verification of requirements overarching model levels is reduced to many small, local refinement proofs by the compositionality of the refinement of the FOCUS theory. The global proof obligations are represented in Fig. 8 by chains of refinement relations. This reduces the complexity of finding proofs, encourages reusability, and can increase the productivity of development engineers through parallelization.

3. Traceability

MontiBelle ensures traceability through refinement relationships. Refinement relations can be checked continuously and thus improve traceability compared to unchecked relations. Continuous checking of refinement relations is possible on a local computer, for example, in an environment similar to an IDE. It is also possible to verify the refinement relations in batch mode, for example, in GitLab©CI/CD or Github©Actions. This is particularly relevant when several subsystems are integrated into an overall architecture. Changing a requirement immediately leads to updated obligations to provide new evidence. These proof obligations are automatically encoded in a theorem prover via a code generator. The proof obligations can be checked automatically through the automation of proof seekers and tactics [48].

Because traceability is a key indicator for upward completeness, the same rationale, which MontiBelle uses to increase upward completeness, applies here. In particular, the compositionality of refinements allows one to automatically infer from the complete refinement of individual subsystems to the refinement of entire systems, as illustrated in Fig. 8. Simply put, this increases the value and significance of these individual refinement relationships. Thus, by promoting the methodical use of refinement relations, the MontiBelle methodology increases traceability.

4. Modifiability

Three aspects of modifiability can be differentiated: 1) maintainability, 2) extensibility, and 3) reusability [10]. Each of these aspects is explained individually in the following paragraphs and related to the presented model-driven approach.

The approach improves *maintainability* by ensuring the correctness of changes to requirements resulting from maintenance work. By using MontiBelle, the semantics of the original specification can be compared to the modified specification. It can thus be formally verified that the maintenance work had the intended effect on the system specification. In addition, relationships to other system specifications (e.g., refinement to hardware-specific requirements) can be continuously verified. It is important to note that the approach can verify underspecified system specifications, making even early and relatively vague system specifications testable. The verification results can be used to guide the maintenance work. This enables semantically oriented maintenance with formally validated results.

Extensibility is improved by analogous reasoning. With the presented model-driven approach, extended system specifications can be semantically checked for old and new requirements or compared to the original system specification. The use of underspecification in early development phases enables the model-driven approach to provide proof of correctness or to automate the delivery of counterexamples. For example, let us assume that a system specification was developed from safety requirements. Let us also assume that the system specification is extended in such a way that resources are saved, for example, subsystems could be merged to deliver multiple functions and thus safe costs, or communication channels could be reduced to safe physical connections. Such optimizations may

[‡‡]Name changes can be handled automatically by renaming all references. MontiCore [47] provides a symbol infrastructure for this purpose, which makes it possible to track named references across all model artifacts.

be unsafe, as combining multiple systems could lead to unwanted behavior or the cable paths could reduce fault tolerance. For safe systems, the approach can provide assurances in the form of formal safety proofs. If the extended system specification is not safe, F-IDE can be used to check potential counterexamples. Utilizing underspecification, system optimizations such as those described earlier can thus be modeled and verified at early stages of development.

MontiBelle promotes the *reusability* of models by allowing for the models to be categorized in a refinement hierarchy. More abstract requirements can serve as a starting point for a tree of system specifications developed from them. Each developed system specification can be an independent subset of the specification. This enables the creation (and maintenance; see the aforementioned in the second paragraph of this section) of product families. The hierarchical tree structure enables the reuse of verification artifacts. Suppose a specification for a data transmission system in avionics has been developed (and verified by the presented model-driven approach) from a set of requirements. An overview of the potential development artifacts is depicted in Fig. 9. The requirements specify a fixed maximum tolerance for the transmission delay. The developed specification must adhere to these tolerances, but is still underspecified in terms of the exact delay. Let us assume that a high-end system has been developed based on these specifications, which guarantees a delay of no more than 1 ms. For a less critical application, a less expensive system with a higher delay tolerance could now be developed, based on the same specification. With MontiBelle, it is sufficient to show the correct development of this less expensive system from the intermediate specification, for example, the refinement relation between them. This can reduce the overall costs for the certification of product families.

5. Freedom from Redundancy

The case study DLUF includes an HLR model of a buffer [36] (more abstract than the lower-level requirement) and an LLR model of the buffer. These two models implement the behavior of a buffer in different ways. Changing the behavior or interface of one of the models can lead to the model that depends on it also having to be changed. The MontiBelle methodology is therefore not free of redundancies. However, this problem is dealt with by the specification of the refinement relationship. If redundant information is syntactically inconsistent, F-IDE provides model analysis tools to identify these inconsistencies. If inconsistencies are semantic in nature, the approach allows for automated verification. If this check is successful, the change is classified as verified (see *modifiability*). If it fails, the approach can be used to search for counterexamples regarding the refinement relation. The redundancy in the behavior description is desired and is therefore not a shortcoming of the approach.

6. Controlled Redundancy

Controlled redundancies in the MontiBelle methodology offer the possibility to specify the behavior both descriptively and prescriptively [38]. Descriptive specifications, in the form of HLRs, are represented by so-called history-oriented specifications. The prescriptive specifications, in the form of LLRs, are represented as state-based specifications by automata [49]. These redundancies enable correctness and consistency checks between the specifications, which ultimately increase the correctness of the developed system. Models can also be provided in a reader-specific way through (de)composition. A requirements manager may need a more abstract view of the system and therefore may consider a descriptive specification of a composed system, for example, the HLR model of DLUF in Fig. 4. A software developer for a subsystem, on the other hand, is more interested in the prescriptive, state-based behavior of the respective subsystem, for example, the LLR model of the buffer in Fig. 5.

D. Quality Requirements for a Modeling Notation

Some model-quality indicators go beyond the measurement of individual models and instead evaluate the modeling notation itself. These are described next.

1. Degree of Formalization

The use of the approach increases the degree of formalization in two ways. First, MontiBelleML requires a minimum level of formalization. The interfaces of the systems and subsystems and their interconnections must therefore be specified. It is permitted to adapt the interfaces and structures in further steps. However, a development step is only complete when the structure of the system specification has been defined at this level. Furthermore, behavioral specifications cannot be described informally at will, but must be created in one of three ways. These are the following: history oriented (abstract), state oriented (implementation oriented), and decomposing (structural).

Second, beyond this minimum level, MontiBelleML allows arbitrarily underspecified behavior, which remains the same or is refined with each development step. This increases the specificity, and the solution space becomes smaller.

2. Adequacy for the Application Domain

MontiBelle is language agnostic, at least as far as the concrete syntax is concerned. As shown in [34,35], an intermediate representation of the models is formed, which captures the semantic domain of architectural description languages and behavioral specifications. In the next step, this representation is mapped to FOCUS. A domain with a corresponding domain-specific language (DSL) can therefore be easily supported, provided that the semantic domain is compatible

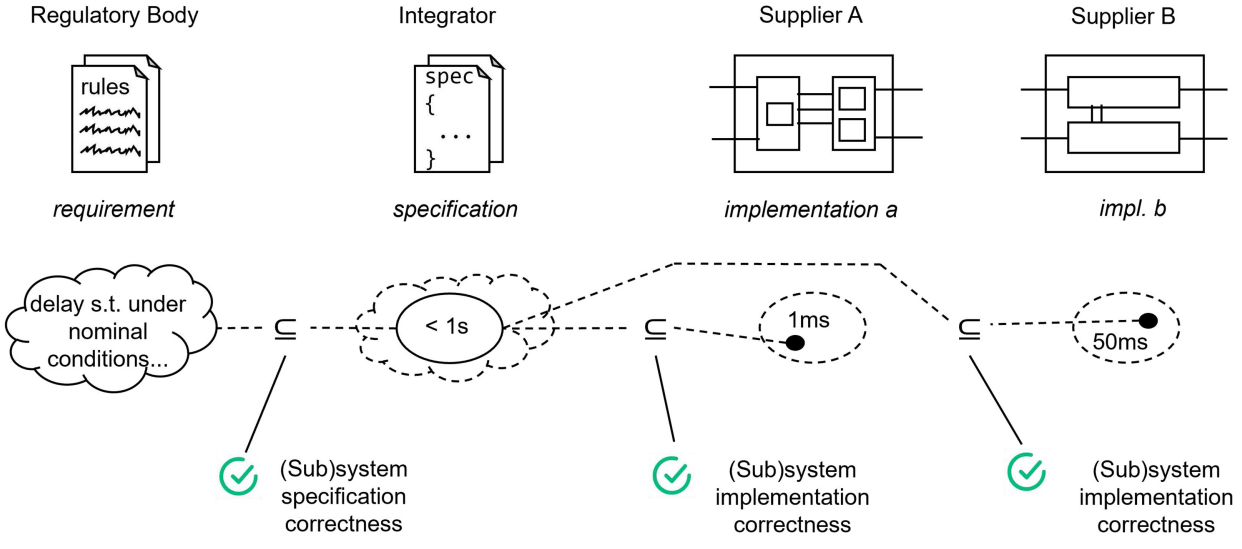

Fig. 9 An overview of the potential development artifacts of an exemplary development of a data transmission system for avionics.

with FOCUS or can be extended by language composition [50]. By adhering to the guidelines for DSLs [51], the adequacy for the application domain can be further increased.

As one of the concrete modeling languages implemented by MontiBelle, MontiBelleML is a profile of SysML v2, the successor to the de facto standard language of systems engineering, SysML v1. SysML v1 is widely used, generally known, and understood. Compared to v1, SysML v2 offers many enhancements and improvements, thanks to a textual representation with the same graphical visualization. SysML v1 will no longer be developed.[§§] Accordingly, we selected SysML v2 as a future-proof modeling language basis. The suitability of MontiBelleML was further demonstrated in the case study, in which HLRs, architecture, and LLRs were modeled.

E. Further Quality Features

The aforementioned set of indicators presented here is not exhaustive. Different domains may require specific quality indicators. In the area of avionics, we consider the following two indicators to be particularly relevant due to the pronounced safety requirements of the domain.

1. *Verifiability*

The approach requires a certain degree of formalization of the models, as described in the corresponding paragraph in Sec. V.D. This ensures verifiability, as machine support is possible. Specifically, the model-driven approach maps the model to its FOCUS semantics via a language-agnostic intermediate model. For this purpose, definitions and theorems are coded in Isabelle and linked to the core definitions of FOCUS. These core definitions of FOCUS were also coded in the theorem prover. This allows for reasoning to be performed on the semantics of the models to handle underspecification and arbitrary runtimes.

The model-driven approach not only achieves verifiability, but also improves it through its development methodology. The use of hierarchies in modeling not only decomposes system requirements (which increases reusability), but also the obligations to provide evidence. This leads to smaller and therefore more automatable proof obligations. The verification of the overall system is ensured by the compositionality of the refinement in FOCUS by design and can be checked automatically.

2. *Transformability*

The model-driven approach achieves model transformability in the following way: First, any data flow and structure modeling language is mapped to a semantically sound foundation. Here, this is achieved by using the SysML v2 profile MontiBelleML. This profile ensures that all models that correspond to the profile are transformable, that is, semantically sound. The transformability then allows for the automated and therefore less error-prone translation of the models into a theorem prover, as explained in the preceding paragraph on *verifiability* of Sec. V.E. The use of the textual representation of SysML v2 also ensures that this transformability holds for all models of the SysML v2 language and is independent of a vendor-specific data structure of the models.

VI. Discussion

Although it can be argued that the presented model-driven approach improves many of the quality indicators, it must be conceded that the main contributions are focused on eight indicators: correctness, consistency, traceability, precision, verifiability, transformability, modularization, and adequacy for the application domain. Some more indicators are beneficially addressed as a result. These indicators are the degree of formalization, semantic adequacy, conceptual integrity, uniformity, conformity, upward completeness, controlled redundancy, and universality.

Representation is not influenced by the MontiBelle, but instead by the chosen modeling language and the associated editor. As MontiBelle is, in general, language agnostic in its core, improving this indicator is left to language designers and tool vendors.

Simplicity is also not at the core of MontiBelle, but is dependent on the domain. Given the highly complex and safety-critical domain of avionics, models are required to be more formal and may appear less simple. It is important, however, to remind oneself of the definition of simplicity, which defines it w.r.t. the additional complexity compared to the required amount due to the problem at hand. This means that seemingly more complex models in an equally complex domain might be simple as a result of this definition. Although the presented approach itself cannot improve simplicity, its ability to modularize and reuse could be leveraged by creating a verified model library. This model library carries additional verification artifacts for the included models. Reuse of avionics-specific building blocks could then simplify existing models.

Language-specific, semantic quality requirements, both internal and horizontal, are hard to quantify absolutely. This is because they largely depend on the capabilities of the modeling language and domain-specific requirements. As MontiBelle is language agnostic at its core, there is no objective case to be made. The presented approach could, however, adapt to domain specificities and languages in the future, precisely because of its language agnosticity.

Consistency and conceptual integrity are another topic highly related to underlying modeling languages, domain-specific concepts, and potentially a library or reusable partial solutions. Again, MontiBelle's contributions here are currently none, but could be increased through the creation of a verified model-library.

Downward completeness is, contrary to upward completeness, not specifically addressed by MontiBelle. The presented approach can formally assure upward completeness as a result of key contributions in correctness, traceability, and verifiability. However, the presented approach passively obtains downward completeness when an executable model is reached. MontiBelle neither requires nor promotes the creation of these executable models and instead leaves it to the user to make that decision. To increase downward completeness, MontiBelle could promote the use of executable models further, for example, by promoting the use of model-based testing, a technique requiring the creation of executable models.

Cohesion as a general quality indicator is largely influenced by modeling language capabilities and domain-specific guidelines. Modern methodologies such as SPES heavily build on concepts such as viewpoints, modeling abstraction layers, and tracing between them. This enables the modeling of tightly related functions at an abstract level, while distributing technical implementations of those related functions at a more concrete level, for example, hardware. The latest release of SysML, SysML v2, brings the necessary capabilities for modeling these concepts. As such, MontiBelleML is well-equipped for future advancements in modeling methodologies and could increase its contributions to cohesion by further developing its own methodology based on SPES and similar methodologies.

Achieving freedom from redundancies, both horizontal and vertical, is not always compatible with compliance w.r.t. specific domains, especially not in the avionics domain. Products as well as processes in avionics are required by certification authorities to be redundant and have failsafes. This is also reflected in the treatment of the DLUF case study, where subcomponents are redundantly described at different abstraction levels to ultimately automatically show correctness. In this regard, the presented approach is always limited by its application domain and driven by the goal of automation, ultimately assisting instead of burdening the engineer.

Finally, modifiability is largely outside the scope of the presented approach. A weak case could be made, where MontiBelle might even hamper modifiability, as already certified systems might not be easily modifiable, because of sunk cost fallacy. Formal verification is inherently costly and hard to automate in the general case. Again, a verified model library could be beneficial to further decrease costs and improve automation. This could gain modifiability of models, ultimately also improving agility.

The taxonomy has been proven useful in a European industrial collaboration [12], although, according to [10], the completeness and full coverage of the quality attributes is not assumed.

[§§]https://www.omgsysml.org/news-articles.htm.

VII. Conclusion

In summary, it can be stated that a model-driven approach with a formal foundation (FOCUS) has positive effects for the development of high-quality models, according to several model-quality indicators listed in Table 1. To this end, the development of a wireless avionics data link using the presented model-driven approach was examined. This showed that a formally sound approach can make a significant contribution to model quality. As mentioned in Sec. I, higher model quality can also improve product quality. Accordingly, the approach is a good candidate for systems engineering, as its positive impact on model-quality attributes (e.g., verifiability) enables the verification and validation of safety according to EUROCAE ED-216, as shown in Fig. 2, especially in early development phases. This is particularly important for the development of mission- and safety-critical avionics systems. To this end, the MontiBelle tool also offers comprehensive automation to support development engineers. Compared to alternative approaches based on different foundations, the mode-driven approach leverages the compositionality of FOCUS to reduce verification complexity, the current version of Systems Modeling Language (SysML) to allow simple usage by system engineers, and a methodology based on EUROCAE ED-216 for certification.

Although this article has shown that the presented mode-driven approach is suitable for increasing model quality and thus product quality, this section presents future work, which is needed before the approach could be applied in practice: To further reduce modeling redundancy, it is conceivable to make further use of the extensive parameterization options of SysML v2. In particular, SysML offers the possibility to refer to model elements by references instead of repeating them. The extent to which building blocks can be reused in the form of constraints needs to be evaluated. In connection with the preceding aspect, it is planned to further increase the degree of automation. A possible next step would be to use the aforementioned references, making it easier to check the correctness of decomposed systems. Crucially, references to the same constraints could simplify the proof of equivalence or implication between constraints. As mentioned earlier, such an approach can further manage or lower redundancy, which further increases model quality. Additionally, building a library of standard model elements with associated certification artifacts seems to be a relevant aspect. MontiBelle benefits from the fact that the underlying formalism FOCUS already pursues precisely this idea and was, among other reasons, selected for this reason. The reuse of development artifacts in an avionics development process could offer opportunities for more economical development. The integration of the continuous verification technique presented in this article into a continuous tool chain in the avionics development processes could be a step toward this goal and increase efficiency. Finally, it is planned to evaluate the approach on models of systems in productive development. This will determine the suitability of the model-driven approach for very extensive system models and may result in the need for further adaptations of the presented approach.

Acknowledgment

The authors wish to thank the German Federal Ministry for Economic Affairs and Climate Action, AMoBaCoD-Project (Grant No. 20X2201C).

References

[1] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M., "Metrics and Laws of Software Evolution—The Nineties View," *Proceedings Fourth International Software Metrics Symposium*, Inst. of Electrical and Electronics Engineers, New York, 1997, pp. 20–32. https://doi.org/10.1109/METRIC.1997.637156

[2] Galster, M., Zdun, U., Weyns, D., Rabiser, R., Zhang, B., Goedicke, M., and Perrouin, G., "Variability and Complexity in Software Design: Towards a Research Agenda," *ACM SIGSOFT Software Engineering Notes*, Vol. 41, No. 6, 2017, pp. 27–30. https://doi.org/10.1145/3011286.3011291

[3] Broy, M., and Steinbrüggen, R., *Modellbildung in der Informatik*, Springer-Verlag, Berlin, 2004.

[4] Broy, M., *Logische und Methodische Grundlagen der Programm- und Systementwicklung*, Springer Fachmedien Wiesbaden GmbH, Wiesbaden, Germany, 2019. https://doi.org/10.1007/978-3-658-26302-7

[5] Broy, M., *Logische und Methodische Grundlagen der Entwicklung verteilter Systeme*, Springer-Verlag GmbH, Berlin, 2023. https://doi.org/10.1007/978-3-662-67317-1

[6] Stachowiak, H., *Allgemeine Modelltheorie*, Springer, Berlin, 1973.

[7] Rumpe, B., Schoenmakers, M., Radermacher, A., and Schürr, A., "UML + ROOM as a Standard ADL?" *Proceedings Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'99) (Cat. No. PR00434)*, Inst. of Electrical and Electronics Engineers, New York, 1999, pp. 43–53.

[8] Torens, C., "Safety Versus Security in Aviation, Comparing DO-178C with Security Standards," *AIAA Scitech 2020 Forum*, AIAA Paper 2020-0242, 2020. https://doi.org/10.2514/6.2020-0242

[9] Friedenthal, S., and Seidewitz, E., "A Preview of the Next Generation System Modeling Language (SysML v2)," *Project Performance International's Systems Engineering News Journal*, Vol. 95, Nov. 2020.

[10] Fieber, F., Huhn, M., and Rumpe, B., "Modellqualität Als Indikator für Softwarequalität: Eine Taxonomie," *Informatik-Spektrum*, Vol. 31, No. 5, 2008, pp. 408–424. https://doi.org/10.1007/s00287-008-0279-4

[11] "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems," International Electrotechnical Commission TR XXXX, Geneva, 2010.

[12] Arendt, T., Kranz, S., Mantz, F., Regnat, N., and Taentzer, G., "Towards Syntactical Model Quality Assurance in Industrial Software Development: Process Definition and Tool Support," *Software Engineering 2011—Fachtagung des GI-Fachbereichs Softwaretechnik*, Gesellschaft für Informatik e.V, Bonn, Germany, 2011, pp. 63–74.

[13] Arendt, T., "Qualitätssicherung von Softwaremodellen—Ein strukturierter Qualitätssicherungsprozess unterstützt durch eine flexible Werkzeugumgebung innerhalb des Eclipse Modeling Project," Ph.D. Thesis, Philipps-Universität Marburg, Marburg, Germany, Jan. 2014. https://doi.org/10.17192/z2014.0357

[14] Schmedding, D., and Vasileva, A., "Reviews—ein Instrument zur Qualitätsverbesserung von UML-Diagrammen," *Tagungsband des 15. Workshops "Software Engineering im Unterricht der Hochschulen"*, Vol. 1790, CEUR Workshop Proceedings, Aachen, Germany, Feb. 2017, pp. 8–19.

[15] Schnieder, E., and Schnieder, L., *Verkehrssicherheit: Maße und Modelle, Methoden und Maßnahmen für den Straßen- und Schienenverkehr, VDI-Buch*, Springer, Berlin, 2013. https://doi.org/10.1007/978-3-540-71033-2

[16] Kausch, H., Pfeiffer, M., Raco, D., Rumpe, B., and Schweiger, A., "Enhancing System-Model Quality: Evaluation of the MontiBelle Approach with the Avionics Case Study on a Data Link Uplink Feed System," *SE 2024—Companion*, Gesellschaft für Informatik e.V., Bonn, Germany, 2024, pp. 119–138. https://doi.org/10.18420/sw2024-ws_09

[17] Reisig, W., *Petri Nets: An Introduction*, Springer, Berlin, 1985. https://doi.org/10.1007/978-3-642-69968-9

[18] Grönniger, H., Krahn, H., Pinkernell, C., and Rumpe, B., "Modeling Variants of Automotive Systems Using Views," *Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen*, Technische Universität Braunschweig, Braunschweig, Germany, 2008, pp. 76–89.

[19] Pohl, K., Hönninger, H., Achatz, R., and Broy, M., *Model-Based Engineering of Embedded Systems*, Springer, Berlin, 2012. https://doi.org/10.1007/978-3-642-34614-9

[20] Pohl, K., Hönninger, H., Daembkes, H., and Broy, M. (eds.), *Advanced Model-Based Engineering of Embedded Systems*, Springer, Cham, 2016. https://doi.org/10.1007/978-3-319-48003-9

[21] Böhm, W., Broy, M., Klein, C., Pohl, K., Rumpe, B., and Schröck, S., *Model-Based Engineering of Collaborative Embedded Systems*, Springer, Berlin, 2021.

[22] Broy, M., and Stølen, K., *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*, Springer, New York. 2001.

[23] Broy, M., and Rumpe, B., "Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung," *Informatik-Spektrum*, Vol. 30, No. 1, 2007, pp. 3–18. https://doi.org/10.1007/s00287-006-0124-6

[24] Kausch, H., Pfeiffer, M., Raco, D., and Rumpe, B., "MontiBelle—Toolbox for a Model-Based Development and Verification of Distributed Critical Systems for Compliance with Functional Safety," *AIAA Scitech 2020 Forum*, AIAA Paper 2020-0671, 2020. https://doi.org/10.2514/6.2020-0671

[25] "ED-216—Formal Methods supplement to ED-12C and ED-109A," European Organization for Civil Aviation Equipment TR ED-216, Saint-Denis, France, Jan. 2012.

[26] Thomke, S., and Fujimoto, T., "The Effect of "Front-Loading" Problem-Solving on Product Development Performance," *Journal of Product Innovation Management*, Vol. 17, No. 2, 2000, pp. 128–142. https://doi.org/10.1111/1540-5885.1720128

[27] Philipps, J., and Rumpe, B., "Refinement of Information Flow Architectures," *International Conference on Formal Engineering Methods Proceedings*, edited by M. Hinchey, IEEE CS Press, Hiroshima, Japan, 1997, pp. 203–212. https://doi.org/10.1109/ICFEM.1997.630427

[28] Nipkow, T., Paulson, L. C., and Wenzel, M., *Isabelle/HOL:A Proof Assistant for Higher-Order Logic, Lecture Notes in Artificial Intelligence*, Springer, Berlin, 2002. https://doi.org/10.1007/3-540-45949-9

[29] Bansiya, J., and Davis, C., "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, 2002, pp. 4–17. https://doi.org/10.1109/32.979986

[30] Kriebel, S., Raco, D., Rumpe, B., and Stüber, S., "Model-Based Engineering for Avionics: Will Specification and Formal Verification e.g. Based on Broy's Streams Become Feasible?" *Proceedings of the Workshops of the Software Engineering Conference. Workshop on Avionics Systems and Software Engineering (AvioSE'19), CEUR Workshop Proceedings*, edited by S. Krusche, K. Schneider, M. Kuhrmann, R. Heinrich, R. Jung, M. Konersmann, E. Schmieders, S. Helke, I. Schaefer, and A. Vogelsang, et al., Vol. 2308, CEUR Workshop Proceedings, van Hoorn, 2019, pp. 87–94.

[31] Haber, A., Ringert, J. O., and Rumpe, B., "MontiArc—Architectural Modeling of Interactive Distributed and Cyber-Physical Systems," RWTH Aachen Univ. TR AIB-2012-03, Aachen, Germany, Feb. 2012.

[32] Cofer, D., and Miller, S. P., " Formal Methods Case Studies for DO-333," NASA CR 2014-218244, 2014.

[33] Kausch, H., Pfeiffer, M., Raco, D., and Rumpe, B., "An Approach for Logic-Based Knowledge Representation and Automated Reasoning over Underspecification and Refinement in Safety-Critical Cyber-Physical Systems," *Combined Proceedings of the Workshops at Software Engineering*, edited by R. Hebig, and R. Heinrich, Vol. 2581, CEUR Workshop Proceedings, Aachen, Germany, 2020, p. 8.

[34] Kausch, H., Pfeiffer, M., Raco, D., Rumpe, B., Götz, L. Linsbauer, Schaefer, I., and Wortmann, A., "Model-Based Design of Correct Safety-Critical Systems Using Dataflow Languages on the Example of SysML Architecture and Behavior Diagrams," *Proceedings of the Software Engineering 2021 Satellite Events*, edited by S. Götz, L. Linsbauer, I. Schaefer, and A. Wortmann, Vol. 2814, CEUR Workshop Proceedings, Aachen, Germany, 2021, p. 22.

[35] Kausch, H., Michael, J., Pfeiffer, M., Raco, D., Rumpe, B., and Schweiger, A., "Model-Based Development and Logical AI for Secure and Safe Avionics Systems: A Verification Framework for SysML Behavior Specifications," *Aerospace Europe Conference 2021 (AEC 2021), Council of European Aerospace Societies (CEAS)*, Warsaw Univ. of Technology, Warsaw, Poland, 2021, p. 9.

[36] Kausch, H., Pfeiffer, M., Raco, D., Rumpe, B., and Schweiger, A., "Correct and Sustainable Development Using Model-Based Engineering and Formal Methods," *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, Inst. of Electrical and Electronics Engineers, New York, 2022, pp. 1–8. https://doi.org/10.1109/DASC55683.2022.9925819

[37] Kausch, H., Pfeiffer, M., Raco, D., Rumpe, B., and Schweiger, A., "Model-Driven Development for Functional Correctness of Avionics Systems: A Verification Framework for SysML Specifications," *CEAS Aeronautical Journal*, Vol. 15, No. 4, 2024. https://doi.org/10.1007/s13272-024-00762-6

[38] Kausch, H., Pfeiffer, M., Raco, D., Rath, A., Rumpe, B., and Schweiger, A., A Theory for Event-Driven Specifications Using Focus and MontiArc on the Example of a Data Link Uplink Feed System," *Software Engineering 2023 Workshops*, edited by I. Groher, and T. Vogel, Gesellschaft für Informatik e.V, Bonn, Germany, 2023, pp. 169–188.

[39] Kausch, H., Koppes, K., Netz, L., O'Brien, P., Pfeiffer, M., Raco, D., Radny, M., Rath, A., Richstein, R., and Rumpe, B., "Applied Model-Based Co-Development For Zero-Emisson Flight Systems Based on SysML," *Proceedings of the Deutscher Luft und Raumfahrt Kongress*, Die Deutsche Gesellschaft für Luft- und Raumfahrt (DGLR) e. V., Lilienthal-Oberth, Germany, 2024, p. 10.

[40] Caspi, P., Pilaud, D., Halbwachs, N., and Plaice, J., "Lustre: A Declarative Language for Programming Synchronous Systems," *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages (POPL)*, Assoc. for Computing Machinery (ACM), New York, 1987, pp. 178–013. https://doi.org/10.1145/41625.41641

[41] Becker, S., Koziolek, H., and Reussner, R., "The Palladio Component Model for Model-Driven Performance Prediction," *Journal of Systems and Software*, Vol. 82, No. 1, 2009, pp. 3–22. https://doi.org/10.1016/j.jss.2008.03.066

[42] Voss, S., and Zverlov, S., "Design Space Exploration in AutoFOCUS 3—An Overview," *IFIP First International Workshop on Design Space Exploration of Cyber-Physical Systems*, edited by V. Mařík, J. M. Lastra, and P. Skobelev, Springer, Berlin, 2014.

[43] Becker, S., Dziwok, S., Gerking, C., Heinzemann, C., Thiele, S., Schäfer, W., Meyer, M., Pohlmann, U., Priesterjahn, C., and Tichy, M., "The MechatronicUML Design Method: Process and Language for Platform-Independent Modeling," Software Engineering Dept., Fraunhofer IEM TR RI-14-337, Paderborn, Germany, March 2014.

[44] Lee, E., "Fundamental Limits of Cyber-Physical Systems Modeling," *ACM Transactions on Cyber-Physical Systems*, Vol. 1, No. 1, 2016, pp. 1–26. https://doi.org/10.1145/2912149

[45] Kounev, S., Rathfelder, C., and Klatt, B., "Modeling of Event-Based Communication in Component-Based Architectures: State-of-the-Art and Future Directions," *Proceedings the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA)*, Elsevier, Amsterdam, The Netherlands, 2012, pp. 3–9.

[46] Kounev, S., Rathfelder, C., and Klatt, B., "Modeling of Event-Based Communication in Component-Based Architectures: State-of-the-Art and Future Directions," *Electronic Notes in Theoretical Computer Science*, Vol. 295, May 2013, pp. 3–9. https://doi.org/10.1016/j.entcs.2013.04.002

[47] Hölldobler, K., Kautz, O., and Rumpe, B., *MontiCore Language Workbench and Library Handbook: Edition 2021*. Aachener Informatik-Berichte, Software Engineering, Shaker Verlag, Band 48, Düren, Germany, 2021.

[48] Bürger, J. C., Kausch, H., Raco, D., Ringert, J. O., Rumpe, B., Stüber, S., and Wiartalla, M., *Towards an Isabelle Theory for Distributed, Interactive Systems—The Untimed Case*, Aachener Informatik Berichte, Software Engineering, Shaker Verlag, Band 45, Düren, Germany, 2020.

[49] Paech, B., and Rumpe, B., "A New Concept of Refinement Used for Behaviour Modelling with Automata," *Proceedings of the Industrial Benefit of Formal Methods (FME'94)*, Springer, Berlin, 1994, pp. 154–174. https://doi.org/10.1007/3-540-58555-9_94

[50] Haber, A., Look, M., Mir Seyed Nazari, P., Navarro Perez, A., Rumpe, B., Völkel, S., and Wortmann, A., "Composition of Heterogeneous Modeling Languages," *Model-Driven Engineering and Software Development, Communications in Computer and Information Science*, Vol. 580, Springer, Berlin, 2015, pp. 45–66. https://doi.org/10.1007/978-3-319-27869-8_3

[51] Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., and Völkel, S., "Design Guidelines for Domain Specific Languages," *Domain-Specific Modeling Workshop (DSM'09)*, Helsinki School of Economics, Espoo, Finland, 2009, pp. 7–13.

E. Atkins
Editor-in-Chief