Enhancing System-model Quality: Evaluation of the MontiBelle Approach with the Avionics Case Study on a Data Link Uplink Feed System

Hendrik Kausch [1], Mathias Pfeiffer [1], Deni Raco [1], Bernhard Rumpe [1], Andreas Schweiger [2]

Abstract: Software quality is often related directly to the quality of the models used throughout the development phases. Assuring model quality can thus be an important aspect for assuring the quality of the final product. Measuring model quality is done via different quality indicators. In this article, we investigate the influence of our holistic systems engineering methodology on model quality. An avionics case study was previously conducted using our methodology. The developed SysML v2 model artifacts are evaluated in this paper regarding internal and external model quality, as well as model notation quality. In total, the positive impact on 26 model quality indicators from our previous work is argued. These indicators are divided into intra-model (single artifact) quality indicators and inter-model (across model artifact) quality indicators. The inter-model quality indicators are further classified into indicators for models at the same granularity level (horizontal) and across several granularity levels (vertical). Multiple quality indicators are positively affected by the modeling language's capabilities and the underlying mathematical semantics. Other indicators depend on methodological guidelines that steer the engineering process. The evaluation of model-quality properties leads towards maturing a holistic systems engineering methodology that facilitates high model quality and thus indicates high product quality.

Keywords: Model Quality; Design Methodology; Theorem Prover; Formal Verification

1 Introduction

1.1 Model Development in Computer Science

It can be observed that the size of software increases continuously [Le97]. Additionally, the degree of its complexity grows. These developments make the full understanding of software systems more difficult. In order to maintain control over such complex software systems, adequate mechanisms are necessary. These include, e.g., approaches according to the principle divide et impera (Latin) or a suitable model usage. Model building represents the essential foundation of the tasks of computer science [BS04, Br19, Br23], since the representable, processable, storable and transferable representation (syntax) of information must always be subjected to interpretation in order to derive the intended semantics. To achieve this, syntactic representations abstractly represent the object of consideration, in order to represent the characteristics relevant for the application purpose. Accordingly, models usually exhibit the following properties [St73]:

[1] RWTH Aachen University, Chair for Software Engineering, Aachen, Germany
[2] Airbus Defence and Space GmbH, Manching, Germany

doi:10.18420/sw2024-ws_09

[KPR+24] H. Kausch, M. Pfeiffer, D. Raco, B. Rumpe, A. Schweiger:
Enhancing System-model Quality: Evaluation of the MontiBelle Approach
with the Avionics Case Study on a Data Link Uplink Feed System.
In: SE 2024 - Companion, pp. 119–138, Gesellschaft für Informatik e.V., Feb. 2

- **Mapping**: Models are always related to an original, which they depict.
- **Reduction**: Models generally do not capture all attributes of the original represented by them, but abstract from them as required for the respective purpose or use case.
- **Pragmatism**: Models are not uniquely mapped to their originals per se, rather they fulfill their substitution function for certain subjects under certain restrictions (purpose and boundary conditions of the model construction). Thus models are subject to variability with respect to the mentioned aspects.

Furthermore, models can be differentiated according to their purpose:

- A **descriptive** model describes an original for understanding or analysis (e.g., a city map).
- A **prescriptive** model describes the way of making of an original (e.g., a construction plan).

In addition, significant value is placed on the **visualization** of models. This is supported by the wide acceptance of semi-formal modeling languages in both the academic and industrial domains. Examples are UML for software or SysML for systems development. The introduction of such models alone will not be able to make the desired contribution to mastering complexity, unless the models have the quality properties relevant to the development project at hand. A model is the most important artifact of model-based development, and its quality thus contributes decisively to the success of the project [FHR08]. For this reason, quality assurance guidelines such as those in the IEC 61508 standard [IE10] rightly demand adequate quality of models in the development process. In particular, quality requirements for models also include requirements for the modeling notation in use. In Tab. 1, quality properties for models according to [FHR08] are summarized and explained. This taxonomy forms the basis for the evaluation (see Sect. 3) of the MontiBelle approach for model-based development and formal verification presented in Sect. 2.

Tab. 1: Overview of quality properties for models [FHR08, p. 416-420] with extensions of vertical quality properties redundancy and controlled redundancy, as well as additional quality properties verifiability and transformability

Intra-model quality properties	
Representation	Representation measures how easily information can cognitively be comprehended. A better representation yields better understanding. Aesthetically pleasing and well-structured models are easier to understand.
Precision	Precision measures whether all relevant properties of the modeled system have been captured. Precision addresses the reduction of information as a result of modeling.
Universality	Universality measures whether only relevant details are modeled. A low universality could result from unnecessarily fixing platform-specific details in early design phases and could, e.g., induce a complex solution in hardware later on.
Simplicity	Simplicity measures whether relevant details are not modeled any more complex than necessary. Simplicity can be increased without losing any information content, e.g., by reformulating, restructuring, and using abstraction.
Semantical adequacy	Semantical adequacy expresses the suitability of a model to purposefully represent desired information. For example, entity-relationship-models are well suited to represent entities and their relations. They are less suited to model the behavior of software components.

Continued on next page

Tab. 1 – Continued from previous page

Consistency	Internal Consistency measures the absence of errors within a single model. A typical consistency check verifies that variables are defined before their use.
Conceptual integrity or uniformity	Conceptual integrity or uniformity aims at providing similar solutions to similar problems. This quality indicator measures the similarity of modeled solutions within single models. Conceptual integrity can be achieved by applying repeatable rules, patterns, and principles.
Conformance	Conformance measures the level of application of standards and norms to models. Conceptual integrity can thus be a consequence of conformance, assuming the standards and norms have relatively tight boundaries. Typically, project specific guidelines are required to achieve conceptual integrity from conformance.
Language-specific, semantical quality properties	These quality properties include any additional quality indicators that the specific modeling language provides. Examples include completeness of state charts or liveness in Petri nets [Re85].
Horizontal inter-model quality properties	
Consistency, conceptual integrity, language-specific, semantical quality properties	Intra-model quality properties in the inter-model perspective. For example, matching interfaces, i.e., the description of visibility along with management and transfer of interface elements, are necessary.
Downward completeness	Downward complete models must contain all necessary information for developing model artifacts of the next development step in the model-chain.
Cohesion	For cohesive models, interrelated parts and facts must be modeled in closely related model artifacts.
Modularity	Modular models must represent single aspects to facilitate re-usage of model artifacts.
Freedom from redundancies	Even though complete redundancy-free modeling is not possible or desired (see controlled redundancy), redundancy between models must be minimized.
Controlled redundancy	Allowing purposeful redundancies in models can be leveraged , e.g., for different views of system parts or facts [Gr08].
Vertical inter-model quality properties	
Correctness	Correctness measures whether a model implements requirements from previous artifacts in the model chain rightly. This could include that requirements of previous stages are developed towards a correct implementation.
Downward correctness	A granularity level of models is downward correct, if all requirements of this level's artifacts are refined in subsequent granularity levels.
Upward completeness	Upwards completeness measures how complete the requirements of a previous development phase were adhered to. Upwards completeness is highly related to correctness, as only correct derivations may result in completeness. However, correctness does not induce completeness. Consider requirements of a previous stage to be implemented in multiple new models. Correctness can be measured for each new model individually. In contrasct, completeness requires consideration of the whole model set.

Continued on next page

Tab. 1 – Continued from previous page

Traceability	Traceability measures the ability to track which information from one model was used to create (parts of) a new model in a subsequent development phase.
Modifiability	Modifiability measures the ability to maintain (for correcting and for evolving), extend, and re-use models or model elements.
Freedom from redundancies	To achieve freedom from redundancies over subsequent granularity level's artifacts, subsequently developed models should not model previously modeled information again.
Controlled redundancy	Introducing controlled redundancies over multiple granularity levels, i.e., modeling the same properties or behaviors in different ways, allows checking semantical consistency of requirements between successively developed model artifacts.
Quality properties for modeling notation	
Degree of formalization	The degree of formalization measures whether and to which extent the models adhere to a formal language. An increased extent of formalization increases the ability to analyze, simulate, or generate artifacts from models. Later development phases generally require higher extent of formalization.
Adequacy for the application domain	Measures whether and how well a modeling notation can be used to model typical domain concepts. Models should generally be clear and compact, while still maintaining the ability to model even complex domain specific concepts. Non adequate notations typically lead to unnecessarily complex models or even non modelable concepts.
Other model quality properties	
Verifiability	Verifiability determines if and how well modeled properties can be verified regarding their correctness. Verifiability requires a certain extent of formalization, as models and their properties need clear semantics for verification to take place. A special case of verification is testing, which requires an executable model.
Transformability	Transformability measures whether and how easily models can be processed and transformed, typically but not necessarily by machines. Transformations can be used to derive simulations or other (formal) analysis.

1.2 Contribution

As described in Sect. 1.1 model-based development provides an indispensable means for mastering increasing complexity of software in general and in particular in the avionics domain. This in turn has a positive impact on the product quality. Since the product quality is determined by the models' quality, we have to provide guidance on how to reach said quality.

Bansiya and Davis [BD02] developed an approach for quantitative assessment of the design properties in object-oriented designs. From the corresponding metrics they calculate the high-level quality attributes. However, the approach is valid for object-oriented systems, while model-based development is the more general approach, which we are targeting in this contribution.

In addition, since some system properties cannot be verified exhaustively by testing only and others are amenable to verification only by the deployment of formal methods, the model-based approach needs to cover the integration of formal methods, as well. To the best of our knowledge, no such method has been presented yet. The contribution of this paper is answering the following research questions:

- **RQ (1)**: Which quality attributes of models are relevant for determining the product's quality?

- **RQ (2)**: To which extent does the selected model-based approach with formal underpinning for precise semantics support the development of models of high quality?

1.3 Structure

RQ (1) has been answered with the proposal of the attributes described in Tab. 1, where these are applicable to any model-based development approach. Sect. 2 covers the MontiBelle approach. This provides the required semantical underpinning via FOCUS, offers a tool chain, includes a development methodology for mitigating the risks of formal verification, and increases the success rates of such a verification. Using an avionics case study in Sect. 3 we evaluate to which extent MontiBelle addresses the identified model quality properties. This answers **RQ (2)**. We conclude in Sect. 4 and describe needs for future development in Sect. 5.

2 MontiBelle Approach

The approach under evaluation in this paper is MontiBelle [Ka20b]. It provides a framework for model-based formal verification. The framework consists of (1) FOCUS [BR07] as a semantic foundation for the interpretation of system models, (2) a methodology for system development, and (3) a tool chain.

2.1 FOCUS

In the formal specification language FOCUS, distributed and interactive systems consist of components that exchange messages over unidirectional channels. The semantics of a component is a (set of) stream processing functions, each of which represents one potential behavior. The refinement of a component's behavior is represented by set inclusion ($\subseteq$) between the original and current components' semantics. Concurrency is represented by an appropriate composition operator that connects channels. The most important reason for using FOCUS is the property that refinement is fully compositional [BR07, Ka20b]. This means that if a system has been decomposed and the parts were refined separately, then the reassembled refined parts are by design a correct refinement of the system before its refinement.

2.2 Methodology

The MontiBelle methodology is a development approach derived from SPES [Bö21] and SpesML [Ge23]. It fulfills the requirements of EUROCAE ED-216, which describes the use of formal verification in the development of software systems for the avionics domain. The MontiBelle methodology aims at providing users with the necessary guidelines to intuitively and successfully apply formal verification to system engineering tasks. MontiBelle uses model-driven concepts to enable abstraction where possible. At the same time, formal specification techniques allow fine-grained control over system specifications where necessary. The MontiBelle approach covers the early stages of development, as well. As such, it provides means to ensure conformance, consistency, verifiability, and traceability between system requirements (SRs), high-level requirements (HLRs), and software architecture design and low-level requirements (LLRs).

First, typically informal SRs are formalized in HLRs as declarative specifications over communication histories. We use input-output relations in assumption-guarantee-style[3]. Designing a system is, in accordance with EUROCAE ED-216, divided into two activities: deriving an architecture and developing LLRs. The MontiBelle approach proposes to decompose HLRs into communication architectures of more detailed and specialized HLRs, until a fine-grained enough architecture is reached. When refining declarative specifications to either other declarative specifications or to architectures [PR97], then meeting the resulting proof obligation typically entails showing the implication between (potentially multiple and coupled) logic predicates. The compositionality of refinement in FOCUS and MontiBelle's toolchain automates the proof-finding. The LLRs are formalized using prescriptive models. MontiBelle suggests event-driven automata [PR94, Ka23] to model LLRs as they are implementation-oriented and specifically fitting for software-intensive systems. One of their most important properties is the fact that they are implementable by construction. Lastly, architecture and LLRs are combined into a system design. This is achieved by refining the final HLR architecture to an equally structured architecture composed from LLR. The proof for correctness of the refinements follows directly from the compositionality of FOCUS and is fully automated.

2.3 Avionics Case Study

The development of an avionics data link was previously conducted using the MontiBelle approach [Ka22, Ka23]. This Data Link Uplink Feed (DLUF) system is representative for software systems in the avionics domain and will be used to evaluate the claim of increased model quality. The DLUF system should enable components using a wireless connection (e.g. between an Unmanned Aerial Vehicle (UAV) and its ground station) to transfer prioritized data packets. The objective was to develop a system that adheres to a set of 18 system requirements. It required to formally verify (instead of just demonstrating the correct functionality with non exhaustive tests), that these properties hold for the overall system (instead of just subsystems) in every scenario (instead of just best case scenarios). One of the requirements, the non-starvation (or liveness) property, required the use of formal methods to ensure a correct DLUF system. This property could not be tested for, as it required checking an unknown and potentially infinitely long time frame. It had to hold for the overall system and could not be sufficiently verified by only checking properties of the system's parts, but required the integration of all artifacts into a single coherent claim.

According to the MontiBelle methodology, the non-starvation requirement was encoded in a descriptive model using the textual notation of SysML v2. The specification is contained in a SysML part definition. The interface is modeled using port usages of a custom type called Packets. The HLR shown in List. 1 is expressed using a (descriptive) requirement which groups four SysML constraint usages, three of which as assumptions and the last as guarantee, designated with the *require* keyword.

[3] A style of specification, that asserts certain properties (the guarantees), if certain preconditions (the assumptions), hold.

```
part def DLUF_HLR1 {
  port input: ~Packets[4]; port output: Packets[4];

  satisfy requirement 'non-starvation' {
    assumes 'infinitely long timeframe' {
      ∀i∈{1,2,3,4}. input[i].length() = ∞
    }
    assumes 'message for each interval' {
      ∀i ∈ {1,2,3,4}, t:nat: input[i].atTime(t).length() > 0
    }
    assumes 'individual packet size below max. capacity' {
      ∀i ∈ {1,2,3,4}: ∀v ∈ input[i].values(): v < maxCap[i]
    }
    require 'infinitely many outputs on all channels' {
      ∀i ∈ {1,2,3,4}: output[i].messages().length() = ∞
} } }
```

List. 1: Descriptive 'non-starvation' HLR formally modeled in SysML v2's textual notation.

From the HLR, we developed fine grained specifications using decomposition. All subsystem specifications at this stage are described using descriptive models similar to the one in List. 1. This approach is in accordance with EUROCAE ED-216 where from System Requirememts, HLRs are developed. Traceability is achieved using a derivative of the general SysML specialization relation called refinement, denoted by the *refines* keyword. Once we reached a suitable decomposition level, we developed prescriptive (LLR) models for each non decomposed (atomic) subsystem. The descriptive models are traced to their prescriptive counterparts using refinement relations. The LLRs are eight total subsystems, four buffering components and four capacity gates. A prescriptive buffer specification is shown in Fig. 1, using SysML v2's graphical representation for better readability:

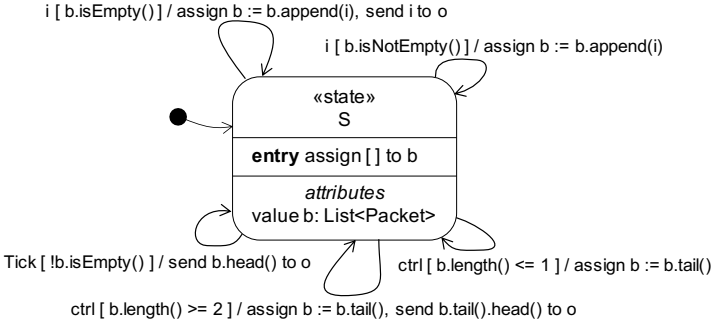

Fig. 1: SysML v2 graphical representation of the prescriptive Buffer model.

The single state contains a list of $Packets$, modeled using SysML v2's attributes. The list stores incoming messages. The transitions handle behavior, i.e., processing incoming messages (top), incoming control directives (bottom right and bottom middle), as well as time passing. Similarly, the capacity gate is shown in Fig. 2.

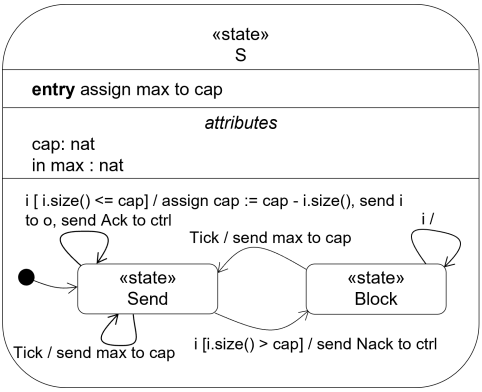

Fig. 2: SysML v2 graphical representation representation of the prescriptive capacity gates.

The capacity gate has two internal states, `Send` and *Block*. The current state shows, whether the current time slice has remaining capacity to send messages. As long as remaining capacity, modeled as the attribute *cap*, is sufficient, messages are forwarded. Once the capacity threshold is reached, messages are blocked until time passes. A control message $Nack$ is sent to inform the buffer component about the blockage. Operation continues, once capacity becomes available. The prescriptive models are assembled into an LLR of the entire system via composition. There are four scheduler subsystems of buffer and capacity gates. A graphical representation of the first subsystem is shown in Fig. 3:

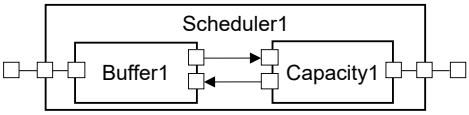

Fig. 3: Graphical representation of a scheduler subsystem from buffer and capacity gate.

The method of decomposing descriptive models and refining them to prescriptive models leads to a tree-structure that describes both the development path, as well as the certification artifacts (proof obligations) required for verification. As part of the case study, all these proof obligations where met in an automated way. Fig. 5 visualizes the tracing relation tree for the DLUF case study.

2.4 Tool Chain

Models of a SysML v2 profile named MontiBelleML describe systems of different granularity or abstraction levels. Between the systems at different abstraction levels, modeled refinement relations indicate proof obligations. A generator translates these models and refinement relations into the syntax of a theorem prover, Isabelle [NPW02]. Isabelle enables machine-assisted and automated proof search. In particular, we use the generated theories together with the general theories (FOCUS encodings) to perform formal verification of the refinement relations. The verification can function autonomously using automation scripts. The result consists of machine-verifiable certificates of correctness or, in cases of errors, counterexamples. The tool chain is depicted in Fig. 4.

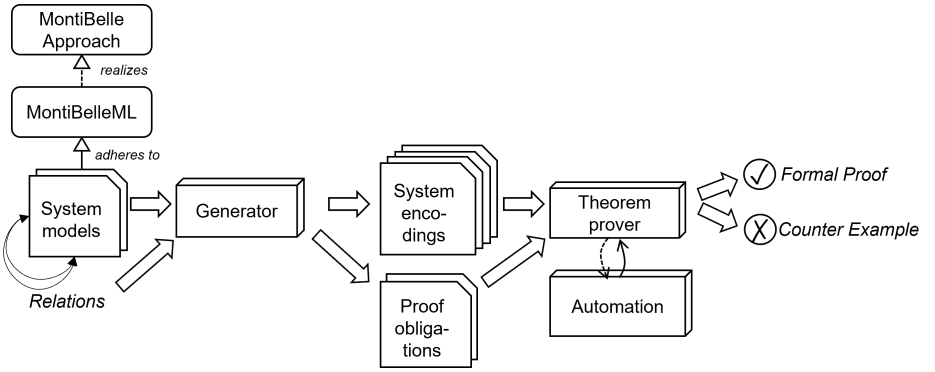

Fig. 4: MontiBelle tool chain overview.

2.5 Related Work

The MontiBelle approach has been applied to case studies from domains such as automotive and aerospace. In [Kr19] time-synchronous modeling was evaluated on an automotive case study. The Isabelle backend was connected to the frontend modeling language MontiArc from the chair of Software Engineering at RWTH Aachen University. MontiArc and a synchronous time model were used again in [Ka20b] for verifying properties of a deterministic pilot flying system from a NASA case study [CM14]. A non deterministic variant of the pilot flying system was refined step by step and proven correct in [Ka20a] using MontiArc and its synchronous time model. Next, SysML was used as frontend to specify the pilot flying system in the synchronous time model [Ka21b], which is more commonly used for hardware modeling. An event-driven modeling approach for distributed software applications was applied in [Ka21a], where the pilot flying system was modeled again using SysML. Using event-driven modeling, another case study from the aerospace domain, the DLUF system, was modeled with SysML and verified by a mapping to Isabelle [Ka22]. Finally, in [Ka23] the DLUF case study was also modeled in MontiArc, where the foundations of event-driven processing using FOCUS were also elaborated.

Formalisms such as Communicating Sequential Processes (CSP) ([Ho85], as used in e.g. [ML09]), Calculus of Communicating Systems (CCS) [Mi82], π-calculus [Pa01], Ptolemy [Le16], Temporal Logic of Actions (TLA) [AL94], Petri nets [Re85] or FOCUS [BR07] are usually used as mathematical underpinning for reasoning. They are typically chosen, because they support non determinism, underspecification, and a notion of behavioral refinement. They further enable the treatment of time-sensitive specifications and hierarchical decomposition. In particular, decomposition is badly needed in general, otherwise the verification of a complex atomic component can quickly become unfeasible because of the computational effort caused by state explosion. This in turn requires compositional[4] verification, which is provided by FOCUS.

Modeling languages can be used to abstract from the complexity of the mathematical formalisms. A number of modeling languages such as Esterel [Be00] or Lustre [Ca87] (and its dialect SCADE) have been created for the development of reactive systems. They are, however, rather suited for the description of hardware systems due to their time-synchronous paradigm. Further methodologies and accompanying tools for specifying distributed systems have been developed, such as the Palladio

[4] Compositionality is introduced by Carnab as *Frege's principle* [Ca47, p. 120-121].

Component Model [BKR09], MechatronicUML [Dz16], AutoFocus [VZ14] or Ptolemy [Le16]. Neither of them supports event-driven specifications or the latest version of SysML, an industry proven and approved modeling language. We chose SysML [Sy23], because it is prominently used in the aerospace and automotive industry for systems engineering.

Lastly, integrating formal verification, particularly deductive methods and modeling languages is not new. The modeling language RSML^{-e} has been combined with the theorem prover PVS [RJH03] via a code generator similarly to our approach. However, the modeling paradigm is synchronous and not event-driven. The modeling language is not an industry standard such as SysML. There exists no automations, as the proving process is manual.

3 Evaluation

The following sections evaluate the MontiBelle approach in regards to our previously introduced quality indicators. We will determine what quality properties are covered by the MontiBelle approach and how so.

3.1 Inner Quality

This section evaluates model quality indicators that are defined for each model individually. Those indicators are called inner quality indicators, because they are defined for single model artifacts. In Sect. 3.2 and Sect. 3.3 quality indicators spanning multiple model artifacts are assessed.

Presentation The MontiBelle methodology suggests the usage of decomposition of specifications. MontiBelle also suggest the use of abstraction and underspecification. Both lead to simpler and more managable model artifacts as demonstrated by the decomposed *Scheduler* in Fig. 3. Furthermore, the MontiBelle approach indirectly increases the cognitive perceptiveness, because instead of proprietary languages, the industry standard language SysML v2 is used. The textual syntax allows adequate structuring by supporting arbitrary formatting. A graphical representation is also offered.

Precision Together with decomposing a system into subsystems and sub-aspects, MontiBelle's stream expressions [Ka22] for specifying requirements allow concisely and separately formulating distinct facts as depicted in List. 1. Besides the signature, there are no additional model artifacts necessary to fully specify systems and their requirements. For close-to-implementation specifications an event-driven, state-based specification technique is provided, which is especially suitable for software-intensive, and time-sensitive systems [KRK13].

Universality MontiBelle suggests strictly history-oriented specifications in the earlier development phases. Besides the system signature and an expected input and output relation, no further assumptions regarding the system's implementation are made. Every implementation that fulfills the input output relation is then conform to the system's specification. Thus, it is ensured that no platform dependencies are introduced. Even in later development phases, MontiBelle recommends abstract (platform independent) state machines as depicted in Fig. 1 and Fig. 2. From these state machines code for any chosen platform that produces output depending on system's states and inputs can be generated by using

a suitable generator. Furthermore, MontiBelle is capable of handling underspecification in models. Underspecification in a specification implies that multiple, potentially infinitely many realizations exist that still fulfill the history- or state-based requirements of the models. Thus, MontiBelle supports variability and management of product-lines.

Simplicity MontiBelle can be actively used to increase simplicity of models. It allows formally verifying semantical compliance of a simpler model (to a previous more complex one), and thus enables correct reformulation and refactoring. Using MontiBelle's underspecification capabilities, refactoring steps permit refinement as well as abstraction of previous models. One example is the refinement and decomposition of the scheduler component into a simpler buffer and capacity component depicted in Fig. 3. Additionally, the approach is at its core syntax agnostic [Ka21b, Ka23]. For domains with assumptions regarding the form of representation of complex properties, a domain-specific adaptation or interchanging the modeling language facilitates domain dependent simplicity (e.g., by requiring a special security keyword to encrypt the transmission, or by using a timing keyword to limit the time delay).

Semantical Adequacy MontiBelle is semantically adequate for developing modern software systems with a focus on safety-critical applications and time-sensitive systems. Since software-systems usually exchange data over a long period of time, the mathematical framework FOCUS is well-suited for depicting such communication histories. The non-starvation requirement of DLUF shows, that liveness properties can be formulated with the MontiBelle approach. Time-critical systems demand specification and analysis of time. For this, timed communication histories and event-driven processing are specially adapted for software-intensive and time-sensitive systems (see [KRK13] for remarks on precision).

Consistency MontiBelle uses the language workbench MontiCore [HKR21] for model processing and consistency analysis. MontiCore offers the possibility to check models for syntax correctness and supports the development of further analyses. Such analyses include checks for compliance with conventions (e.g., naming conventions), reference checks (e.g., existence and visibility of referenced model elements such as system states), and type checks (e.g., type-correct expressions over communication histories, so called stream expressions). The DLUF models are automatically checked in regards to type and interface consistencies.

Conceptual Integrity/Uniformity MontiBelle provides internal conceptual integrity/uniformity through the use of a domain-specific profile for the industry known system modeling language SysML v2. The profile restricts the use of model elements to those that have a semantic foundation in FOCUS, e.g., requirements modeled in List. 1 and state charts modeled in Fig. 1. Of course, it is not sufficient to consider only *internal* conceptual integrity/uniformity. When the subsystems and views of those subsystems come together, the conceptual integrity/uniformity plays a vital role again. The SysML v2 profile therefore is also relevant in Sect. 3.2. Nonetheless, the foundation for this is already laid here.

Conformance MontiBelle requires conformance to the concepts already explained above. 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 relation between models and the verifiability of those models. For example, MontiBelle achieves conformity of models with respect to EUROCAE ED-216 by external, i.e., cross-model, features (compare Sect. 3.2).

Language-specific, Semantical Quality properties This is one of the core competencies of MontiBelle. By building upon a suitable semantical foundation and mapping models into theorem prover code, MontiBelle achieves semantic analyzability of model artifacts, including abstract, history-oriented specifications and underspecification. This is showcased through the verification of the modeled DLUF system in regards to its non-starvation property. Even complex systems can be formally analyzed by leveraging decomposition in FOCUS and its compatibility with refinement. This verification further enhances model quality.

3.2 Horizontal Inter-model Quality Properties

In the following sections, we address inter-model quality indicators at one granularity level. Granularity levels emerge in the iterative system development process as soon as the system is described at a different level of abstraction. According to the MontiBelle methodology, underspecification is decreased across granularity levels, while decomposition is increased.

Consistency, Conceptual Integrity, Language-specific, Semantical Quality Properties These properties can often be assessed only across models. The MontiBelle SysML profile allows system engineers to model refinement dependencies across model artifacts and the modeled system parts. In DLUF, 19 refinement relations (compare Fig. 5) are traced. These refinement relations between models are statically and semantically verifiable. Static verification is a well established model analysis approach. MontiBelle, for example, checks the compatibility of refined interfaces. These checks are made possible by MontiCore's [HKR21] language infrastructure. Semantic verification, however, is only possible with a suitable semantic domain. MontiBelle uses FOCUS to facilitate the formal verification of horizontal inter-model conformity and conceptual integrity. All refinement relations of DLUF were proven in the case study [Ka22].

Downward Completeness Following the MontiBelle methodology, downward completeness is inherently reached, once the current granularity level's specifications fulfill the requirements from the previous level. This is because MontiBelle is able to verify the correct specification of underspecified systems, as shown multiple times for the DLUF case study granularity levels in Fig. 5. Furthermore, MontiBelle promotes the use of decomposition and refinement to gain more granular and eventually less abstract specifications. Fulfillment of previous specifications can be formally verified, as the paragraph about *correctness* explains. Downward completeness is thus highly related to *correctness* and *upward completeness* in the MontiBelle approach.

Cohesion (De)composition supports both cohesion and modularity of models. According to the MontiBelle methodology, interrelated system aspects are modeled as individual subsystems, because this facilitates verification. In the DLUF models this is exemplary shown in the scheduler component. A strong cohesion between the buffer (see Fig. 1) and capacity models (see Fig. 2) exists, since together they specify the scheduler (see Fig. 3). The scheduler model is individually well-defined and can be understood without any other models. Through the methodical application of the MontiBelle approach, closely related system parts of the DLUF system are cohesively modeled.

Modularity (De)composition (see Fig. 5) is arguably even more important for modularity than it is for cohesion. By decomposing a system into multiple subsystems, the system and its models get more

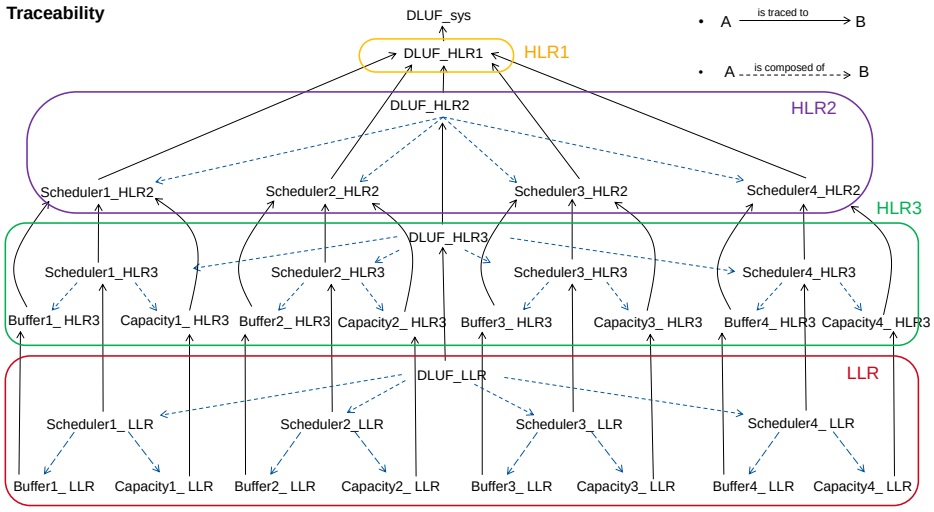

Fig. 5: Hierarchical decomposition of the DLUF case study including tracing and decomposition.

and more modular, since the models of subsystems only model single aspects. This modularity allows independent development into new granularity levels including verifying refinement relations. The independently developed models' interfaces are correct in regard to the system, because the refinement of a subsystem directly implies the refinement of the whole system in Focus. The buffer component in the DLUF system can independently be developed further without having any implication on the scheduler component or the whole DLUF system.

Freedom from Redundancies For systems modeled according to the MontiBelle approach, a granularity level reduces redundancies by importing and re-using models. The scheduler composition in Fig. 3 does not define the buffer's or capacity's interfaces and behavior. Instead, model artifacts are referenced for the composition. Furthermore, additional redundancies are removed in MontiBelle by facilitating parametric models. Only one capacity model is enough to specify the four *different* scheduler components of DLUF [Ka22]. However, completely redundancy free modeling is impossible, e.g., when referencing other model artifacts a model name is used. If this model name is changed, the references to this model must also be changed[5]. Additional well-defined tooling mitigates this problem.

Controlled Redundancy By decomposing the system into individual models, thus introducing controlled redundancy, these models can be independently developed. Model redundancies, when referencing models, allow statically checking type and structural correctness of compositions. Verification coverage is given, since MontiBelle formally verifies the correctness of a granularity level. In DLUF, different system engineers specify and refine different models.

[5] Name changes can automatically be handled by renaming all references. MontiCore [HKR21] provides a symbol infrastructure for this purpose, enabling tracing named references across all model files.

3.3 Vertical Inter-model Quality Properties

In the following sections cross-model and cross-granularity level quality properties are evaluated for the MontiBelle methodology. These quality indicators can be assessed, once model artifacts of systems are iteratively developed.

Correctness For correctness, the requirements from the previous level must have been correctly implemented in the following level. MontiBelle allows the modeling and correctness checking through the refinement relation of the FOCUS theory. This is done in a machine-supported and automated way. For DLUF, all system specifications of all granularity layers and all refinement relations between them are automatically translated into theorem prover syntax. Then all proof obligations are met by automatic solvers and machine checked [Ka22]. In summary, this verified, that the architecture and LLR are correct regarding the top level HLR. MontiBelle not only verifies this quality attribute, but also supports the development of models in this direction. Counterexamples can be found and extracted as test cases, which can facilitate revisions of the models.

Upward Completeness Orthogonal to correctness, this property requires, that all requirements from the level above have been fulfilled. The reasoning therefore follows in large parts that of the previous paragraph. A particularly important feature of MontiBelle's underlying FOCUS theory is the compositionality of refinement. Due to FOCUS, the global consideration of requirements and development steps is reduced to many small, local refinement proofs, which are illustrated in Fig. 5. As a result, the complexity of proof finding is decreased, reusability is stimulated, and engineering productivity can be increased through parallelization.

Traceability MontiBelle ensures traceability through refinement relations. Refinement relations can be continuously checked and thus improve traceability compared to unchecked tracing relations. The continuous verification of the refinement relations is possible locally, for example in an Integrated Development Environment (IDE) like environment. It is also possible to verify the refinement relations in batch mode, for example in GitLab© CI/CD or GitHub© Actions. This is particularly important when multiple subsystems are integrated into an overall architecture. Changing a requirement immediately leads to updated proof obligations. These proof obligations are automatically encoded in a theorem prover via a code generator. The proof obligations can be checked automatically through automation of proof finders and tactics [Bü20]. Furthermore, because traceability is a key indicator for upwards completeness, the same reasoning as to why MontiBelle increases upwards completeness apply here. Particularly, the compositionality of refinements allows us to automatically conclude refinement of complete systems from refinement of individual subsystems as depicted in Fig. 5. Plainly put, this increases the value of those individual refinement relations. By increasing their value and encouraging their methodological usage, MontiBelle increases the traceability.

Modifiability Modifiability can be divided into the three aspects (1) maintainability, (2) extensibility, and (3) reusability [FHR08]. We will argue each aspect individually in the following paragraphs.

MontiBelle improves maintainability by assuring correctness of changes induced by maintenance work to existing requirements. By using MontiBelle, the semantics of the original and the changed specification can be compared. It can thus be formally verified, that the maintenance work had the intended effect on the system specification. Additionally, relations to other system specifications (for

example refinement to hardware- pecific requirements) can continuously be verified, as well. It is important to note, that MontiBelle is able to verify underspecified system specifications, thus can be used to maintain even early and relatively vague system specifications. The verification results can be used to steer the maintenance work. This enables semantic oriented maintenance with formally assured outcome.

Extensibility is improved analogously. With MontiBelle, extended system specifications can be semantically verified regarding old and new requirements or compared to the original system specification. The use of underspecification in early development phases enables proving the correctness of or deliver counterexamples for extended system specifications. As an example, assume a system specification was developed from safety and security constraints. Now assume the system specification is extended in such a way as to save resources. For example, subsystems might be merged to deliver multiple functionalities and thus save costs. Communication channels might be reduced to save cable runs and thus weight. Such optimizations might be unsafe as merging multiple systems might introduce unsafe behavior or reduced cable runs could decrease fault tolerances. For safe systems, MontiBelle is able to provide assurances in the form of formal safety proofs. If the extended system specification is not safe, then MontiBelle can be used to produce and check potential counterexamples. Use of underspecification enables modeling and verification of such optimizations at early development stages.

MontiBelle promotes model reusability by allowing models to be placed in a refinement hierarchy. More abstract requirements can serve as the starting point for a tree of developed system specifications. Each developed system specification might be an independent subset of the specification. This enables the creation (and maintenance, see above) of product families. The hierarchical tree structure allows verification artifacts to be reused. Assume an avionics data transmission system specification was developed from a set of regulatory requirements. An overview of the development artifacts is shown in Fig. 6. The requirements leave some fixed maximum tolerance to the transmission delay. The developed specification must adhere to those tolerances, but is still underspecified regarding the exact delay. A top-of-the-line system was developed from those specifications, having a delay of at most 1 ms. For a less mission critical application, a cheaper system with a higher delay tolerance can now be developed from the same underspecified system specification. With MontiBelle, it suffices to show the correct development of this cheaper system from the intermediary specification. This reduces the overall costs of product families.

Freedom of Redundancies The DLUF system case study includes a buffer HLR [Ka22] and a buffer LLR model. These specify the behavior of a buffer in different abstraction levels. Changing the behavior or interface of one of the models may cause the other model to need to be modified, as well. Thus, the MontiBelle methodology is not free of redundancies. However, this problem is mitigated by specifying the refinement relation. Should redundant information be inconsistent syntactically, MontiBelle provides model analysis tools to find these inconsistencies. If the inconsistencies are of semantical nature, then MontiBelle automatically attempts a re-verification. If this verification is successful, the change is re-verified (see *Modifiability*). If it fails, MontiBelle can provide the system engineer with a counterexample regarding the refinement relation. The redundancy in behavior description is desirable and is therefore not a defect of the methodology.

Controlled Redundancies Controlled redundancies in the MontiBelle approach give the possibility to specify behavior both descriptively and prescriptively [Ka23]. Descriptive specifications, in the form of HLRs, are represented by history-based specifications. Prescriptive specifications (LLRs) are represented as state-based specifications by event automata. These redundancies allow for correctness and consistency checks between the speficiations, ultimately increasing correctness of the

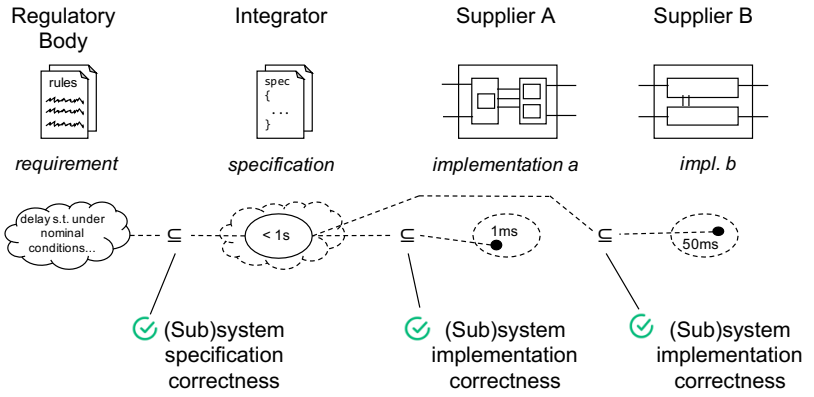

Fig. 6: Exemplary development of an avionics data transmission system with delay constraints. An integrator would have to prove its specifications to be safe regarding some regulatory body's requirements (left). Supplier A would have to prove its system's correctness regarding the integrator's specifications.

developed system. Models can also be provided in a reader-oriented manner by (de)composition. A requirements manager may need a more abstract view of the system and thus considers a descriptive specification of a composed system (e.g., DLUF HLR in List. 1) [Ka22]. A software developer of a specific subsystem meanwhile is more interested in the prescriptive, state-based behavior of the respective subsystem (e.g., buffer LLR in Fig. 1).

3.4 Quality Properties for Modeling Notation

Some model quality indicators go beyond measuring individual models and rather measure the modeling notation itself.

Degree of Formalization Using MontiBelle increases the level of formalization in two ways. First, the SysML v2 profile requires a minimum level of formalization. The interfaces of the systems and subsystems and their interconnections must be specified. It is permitted to adapt the interfaces and structures in later developmemt steps, but one development step is not complete, until the structure of the system specification has been defined at this step and thus granularity level. Further, behavioral specifications cannot be described informally at will, but must be done in one of three ways. These are: history-oriented (abstract), state-oriented (implementation-oriented) and decomposition-oriented (structural). Beyond this minimal degree of formalization, MontiBelle allows arbitrary underspecified behavior, which remains at most the same or is refined with each development step. Thus, after a refinement step the specificity increases and the solution space becomes smaller.

Adequacy for the Application Domain First, MontiBelle is language agnostic in terms of concrete syntax. As shown in [Ka21b, Ka21a], an intermediate representation of models captures the semantic domain of architecture description languages and behavioral specifications. This intermediary

model is then mapped to FOCUS. Thus, a domain coupled with a corresponding domain specific language (DSL) can be easily served, as long as the semantic domain is compatible. As one of the concrete modeling languages realized by MontiBelle, SysML v2 is the successor of the de facto standard language of systems engineering, SysML v1. SysML v1 is widely known, used, and understood worldwide. The successor, SysML v2, offers many enhancements and improvements thanks to its textual representation with the same graphical visualization and is therefore suitable as a language for the domain of systems engineering and was used in the DLUF case study.

3.5 Additional Model Quality Indicators

This list of model quality indicators presented here is not exhaustive. Different domains might come with their own set of quality indicators. We found the two below to be of particular interest for the avionics domain.

Verifiability MontiBelle requires a certain degree of formalization of the models, as described in the corresponding paragraph *degree of formalization*. Through this formalization, one gains verifiability, as it enables machine assisted formal verification. Specifically, MontiBelle maps the model to its FOCUS semantics via an intermediate syntax agnostic model. As a result, definitions and theorems are encoded in a theorem prover and linked to the core definitions of FOCUS, which have also been encoded in the theorem prover. The theorem prover allows reasoning over the semantics of the models. The mathematical underpinning by FOCUS enables capturing underspecification and abstraction. However, MontiBelle not only enables verifiability, but also improves the verifiability through its development methodology. The methodology is based on the SPES methodology and uses granularity layers and decompositon hierarchies in modeling. This enables not only the decomposition of system requirements (which increases reusability), but leads to decomposed proof obligations, as well. This leads to smaller and thus more automatable verification steps. The verification of the overall system is ensured by the compositionality of the refinement in FOCUS. This property holds by construction and can be checked automatically in our Isabelle encoding.

Transformability MontiBelle achieves transformability of models in the following way: First, any data flow and structure modeling language is reduced to semantically well-founded models. In MontiBelle, this is achieved for SysML v2 by a corresponding profile. This profile ensures, that all models that correspond to the profile are transformable, i.e. semantically sound. The transformability then allows the automated and therefore less error-prone translation of the models into a theorem prover. This is detailed in the paragraph about *verifiability*. The textual representation of SysML v2 ensures, that this transformability is given for all SysML v2 models and is independent of any vendor specific tool and data structure of the models. All models of the DLUF case study were transformed and could subsequently be treated by the highest level of formal methods, i.e. deductive reasoning (theorem proving).

4 Conclusion

In summary, modeling along the MontiBelle methodology has a positive impact on all model quality indicators (answering **RQ (02)**) listed in Tab. 1. As argued previously in [FHR08], increasing model quality can also enhance product quality and thus lead to more correct systems. Accordingly, the

methodology is a good candidate for systems engineering, as it allows the verification and validation of system properties in early development phases. This is especially important for the development of highly safety-critical systems of the avionics domain. For this purpose the MontiBelle tool also offers extensive automation to support the development engineers.

5 Outlook

In order to further reduce modeled redundancy, it is conceivable to increase the use of the extensive parameterization possibilities of SysML v2. In particular, SysML provides the possibility to refer to model elements by so-called *references* instead of repeating them. The extent to which, for example, building blocks can be reused in the form of *constraints* is to be evaluated. We also plan to increase the level of automation even more. A possible next step would be to consider the use of the references to make it easier to check decomposed systems for correctness. Crucially, references to the same constraints could simplify the proof of equivalence or implication between constraints. Additionally, the construction of a library of standard model elements with accompanying certification artifacts seems to be an interesting aspect. MontiBelle benefits from the fact, that the underlying formalism FOCUS already follows exactly this idea. Re-use of development artifacts in an avionics development processes could provide opportunities for a more economic development. Integrating the technique of continuous verification introduced in this paper into avionics development processes might be an incremental step towards this goal. Finally, we plan to evaluate our methodology on models of systems in productive development. Some quality indicators are hard to measure and require the measurement of proxy indicators instead. Furthermore, quantitative assertions are hard to prove. These shortcomings should be further investigated.

Bibliography

[AL94] Abadi, Martín; Lamport, Leslie: Open Systems in TLA. In: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing - PODC '94. ACM Press, New York, New York, USA, pp. 81–90, 1994.

[BD02] Bansiya, J.; Davis, C.G.: A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1):4–17, 2002.

[Be00] Berry, Gérard; Bouali, Amar; Fornari, Xavier; Ledinot, Emmanuel; Nassor, Eric; de Simone, Robert: ESTEREL: a formal method applied to avionic software development. Science of Computer Programming, 36(1):5 – 25, 2000.

[BKR09] Becker, Steffen; Koziolek, Heiko; Reussner, Ralf: The Palladio Component Model for Model-Driven Performance Prediction. Journal of Systems and Software, 82:3–22, 01 2009.

[Bö21] Böhm, Wolfgang; Broy, Manfred; Klein, Cornel; Pohl, Klaus; Rumpe, Bernhard; Schröck, Sebastian, eds. Model-Based Engineering of Collaborative Embedded Systems. Springer, Cham, January 2021.

[BR07] Broy, Manfred; Rumpe, Bernhard: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik-Spektrum, 30(1):3–18, Februar 2007.

[Br19] Broy, Manfred: Logische und Methodische Grundlagen der Programm- und Systementwicklung. Springer Fachmedien Wiesbaden GmbH, Wiesbaden, Germany, 2019.

[Br23] Broy, Manfred: Logische und Methodische Grundlagen der Entwicklung verteilter Systeme. Springer-Verlag GmbH, Berlin, Germany, 2023.

[BS04] Broy, Manfred; Steinbrüggen, Ralf: Modellbildung in der Informatik. Springer-Verlag, Berlin, Heidelberg, 2004.

[Bü20] Bürger, Jens Christoph; Kausch, Hendrik; Raco, Deni; Ringert, Jan Oliver; Rumpe, Bernhard; Stüber, Sebastian; Wiartalla, Marc: Towards an Isabelle Theory for Distributed, Interactive Systems – The Untimed Case. Aachener Informatik Berichte, Software Engineering, Band 45. Shaker Verlag, Germany, March 2020.

[Ca47] Carnab, Rudolf: Meaning and Necessity: A Study in Semantics and Modal Logic. The University of Chicago Press, Chicago, IL, USA, 1947.

[Ca87] Caspi, Paul; Pilaud, Daniel; Halbwachs, Nicolas; Plaice, John: Lustre: A Declarative Language for Programming Synchronous Systems. In: POPL. 1987.

[CM14] Cofer, Darren; Miller, Steven P: Formal methods case studies for DO-333. Technical report, 2014.

[Dz16] Dziwok, Stefan; Pohlmann, Uwe; Piskachev, Goran; Schubert, David; Thiele, Sebastian; Gerking, Christopher: The MechatronicUML Design Method: Process and Language for Platform-Independent Modeling. Technical report, Software Engineering Department, Fraunhofer IEM, Paderborn, Germany, December 2016.

[FHR08] Fieber, Florian; Huhn, Michaela; Rumpe, Bernhard: Modellqualität als Indikator für Softwarequalität: eine Taxonomie. Informatik-Spektrum, 31(5):408–424, Oktober 2008.

[Ge23] SpesML Project Site. `https://spesml.github.io`, Accessed: 2023-10-24.

[Gr08] Grönniger, Hans; Krahn, Holger; Pinkernell, Claas; Rumpe, Bernhard: Modeling Variants of Automotive Systems using Views. In: Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen. Informatik Bericht 2008-01. TU Braunschweig, pp. 76–89, 2008.

[HKR21] Hölldobler, Katrin; Kautz, Oliver; Rumpe, Bernhard: MontiCore Language Workbench and Library Handbook: Edition 2021. Aachener Informatik-Berichte, Software Engineering, Band 48. Shaker Verlag, Düren, May 2021.

[Ho85] Hoare, C. A. R.: Communicating Sequential Processes. Prentice Hall International, Englewood Cliffs, N.J., 1985.

[IE10] IEC: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems. Standard, International Electrotechnical Commission, Geneva, 2010.

[Ka20a] Kausch, Hendrik; Pfeiffer, Mathias; Raco, Deni; Rumpe, Bernhard: An Approach for Logic-based Knowledge Representation and Automated Reasoning over Underspecification and Refinement in Safety-Critical Cyber-Physical Systems. In: Combined Proceedings of the Workshops at Software Engineering 2020. CEUR, Online, February 2020.

[Ka20b] Kausch, Hendrik; Pfeiffer, Mathias; Raco, Deni; Rumpe, Bernhard: MontiBelle – Toolbox for a Model-Based Development and Verification of Distributed Critical Systems for Compliance with Functional Safety. In: AIAA Scitech 2020 Forum. American Institute of Aeronautics and Astronautics, Reston, Virgina, January 2020.

[Ka21a] Kausch, Hendrik; Michael, Judith; Pfeiffer, Mathias; Raco, Deni; Rumpe, Bernhard; Schweiger, Andreas: Model-Based Development and Logical AI for Secure and Safe Avionics Systems: A Verification Framework for SysML Behavior Specifications. In: Aerospace Europe Conference 2021 (AEC 2021). Council of European Aerospace Societies (CEAS), Warsaw, Poland, November 2021.

[Ka21b] Kausch, Hendrik; Pfeiffer, Mathias; Raco, Deni; Rumpe, Bernhard: Model-Based Design of Correct Safety-Critical Systems using Dataflow Languages on the Example of SysML Architecture and Behavior Diagrams. In: Proceedings of the Software Engineering 2021 Satellite Events. CEUR, Online, February 2021.

[Ka22] Kausch, Hendrik; Pfeiffer, Mathias; Raco, Deni; Rumpe, Bernhard; Schweiger, Andreas: Correct and Sustainable Development Using Model-based Engineering and Formal Methods. In: 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC). IEEE, USA, September 2022.

[Ka23] Kausch, Hendrik; Pfeiffer, Mathias; Raco, Deni; Rath, Amelie; Rumpe, Bernhard; Schweiger, Andreas: A Theory for Event-Driven Specifications Using Focus and MontiArc on the Example of a Data Link Uplink Feed System. In: Software Engineering 2023 Workshops. Gesellschaft für Informatik e.V., Bonn, pp. 169–188, February 2023.

[Kr19] Kriebel, Stefan; Raco, Deni; Rumpe, Bernhard; Stüber, Sebastian: Model-Based Engineering for Avionics: Will Specification and Formal Verification e.g. Based on Broy's Streams Become Feasible? In: Proceedings of the Workshop on Avionics Systems and Software Engineering (AvioSE'19). CEUR, Online, pp. 87–94, February 2019.

[KRK13] Kounev, Samuel; Rathfelder, Christoph; Klatt, Benjamin: Modeling of Event-based Communication in Component-based Architectures: State-of-the-Art and Future Directions. Electronic Notes in Theoretical Computer Science, 295:3–9, 2013. Proceedings the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA).

[Le97] Lehman, M.M.; Ramil, J.F.; Wernick, P.D.; Perry, D.E.; Turski, W.M.: Metrics and Laws of Software Evolution - The Nineties View. In: Proceedings Fourth International Software Metrics Symposium. pp. 20–32, 1997.

[Le16] Lee, Edward: Fundamental Limits of Cyber-Physical Systems Modeling. ACM Transactions on Cyber-Physical Systems, 1:1–26, 11 2016.

[Mi82] Milner, R.: A Calculus of Communicating Systems. Springer-Verlag, Berlin, Heidelberg, 1982.

[ML09] Murray, Toby; Lowe, Gavin: On Refinement-Closed Security Properties and Nondeterministic Compositions. Electr. Notes Theor. Comput. Sci., 250:49–68, 09 2009.

[NPW02] Nipkow, Tobias; Paulson, Lawrence C.; Wenzel, Markus: Isabelle/HOL: A proof assistant for Higher-Order Logic. Lecture Notes in Artificial Intelligence. Springer, Berlin et al., 2002.

[Pa01] Parrow, Joachim: Handbook of Process Algebra. Elsevier Science, Amsterdam, chapter An Introduction to the Π-Calculus, pp. 479–543, 2001.

[PR94] Paech, Barbara; Rumpe, Bernhard: A new Concept of Refinement used for Behaviour Modelling with Automata. In: Proceedings of the Industrial Benefit of Formal Methods (FME'94). LNCS 873, Springer, Spain, pp. 154–174, 1994.

[PR97] Philipps, Jan; Rumpe, Bernhard: Refinement of Information Flow Architectures. In: ICFEM'97 Proceedings. IEEE CS Press, Hiroshima, Japan, 1997.

[Re85] Reisig, Wolfgang: Petri Nets: An Introduction. Springer, Berlin, Heidelberg, 1985.

[RJH03] Rayadurgam, Sanjai; Joshi, Anjali; Heimdahl, Mats P. E.: Using PVS to Prove Properties of Systems Modelled in a Synchronous Dataflow Language. In: Formal Methods and Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 167–186, 2003.

[St73] Stachowiak, Herbert: Allgemeine Modelltheorie. Springer, 1973.

[Sy23] SysML v2 Official Specification. `https://github.com/Systems-Modeling`, Accessed: 2023-11-29.

[VZ14] Voss, Sebastian; Zverlov, Sergey: Design Space Exploration in AutoFOCUS 3 – An Overview. In: IFIP First International Workshop on Design Space Exploration of Cyber-Physical Systems. 2014.