[KPR+22] H. Kausch, M. Pfeiffer, D. Raco, B. Rumpe, A. Schweiger:
Correct and Sustainable Development Using Model-based Engineering and Formal Methods.
In: 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), IEEE, September 2022.
https://www.se-rwth.de/publications/

Correct and Sustainable Development Using Model-based Engineering and Formal Methods

1st Hendrik Kausch
RWTH Aachen University
Aachen, Germany
kausch@se-rwth.de

2nd Mathias Pfeiffer
RWTH Aachen University
Aachen, Germany
mpfeiffer@se-rwth.de

3rd Deni Raco
RWTH Aachen University
Aachen, Germany
raco@se-rwth.de

4th Bernhard Rumpe
RWTH Aachen University
Aachen, Germany
rumpe@se-rwth.de

5th Andreas Schweiger
Airbus Defence and Space GmbH
Manching, Germany
andreas.schweiger@airbus.com

***Abstract*—Currently, the most wide-spread quality assurance methods in industry are testing and manual reviews, but their costs grow disproportionately with the system size and they cannot achieve exhaustive coverage. To overcome these issues, a model-based verification approach for analyzing cyber-physical systems early in the development phase is presented. A semantics is given to SysML v2 models by a mapping into a (theorem prover encoding of a) data flow formalism. The creation of a SysML profile supporting event-driven and state-based specifications, the encoding of corresponding data flow structures in the theorem prover Isabelle, and a code generator from SysML to Isabelle is shown. The approach is evaluated by formally proving a selected liveness property of a hierarchical system model from the avionics domain. Since liveness properties can be negated only by infinite data sequences and thus cannot be covered exhaustively by testing, this case study suitably demonstrates the added value of our methodology. Finally, this MBSE framework for early design analysis can lead to an optimization of the architecture layout (i.e. reducing the number of required avionics components), which in turn can lead to the reduction of the avionics equipment weight, and thus of the energy consumption, producing less CO_2 emissions.**

***Index Terms*—avionics, formal method, model checking, theorem proving, abstract interpretation, static analysis, SysML, requirements, event-driven automata, non-determinism, underspecification**

I. Introduction

Rising automation during the operation of aircraft drives the complexity of avionics systems. Nowadays, the latter account for over 30 % of the overall aircraft development costs [1]. The effort is mainly generated by the strict safety (RTCA DO-178C) and security (RTCA DO-326A) demands. These are required by the certification authorities (EASA and FAA) and cover the complete development process for both software (RTCA DO-178C) and hardware (RTCA DO-254). The major part of the avionics' development costs is accounted for by the verification phase [2].

RTCA DO-178C's supplements enable the deployment of formal (RTCA DO-333) and model-based methods (RTCA DO-331). The combination of these enables a significant reduction of development costs. Companies in aerospace have successfully adopted formal methods for verifying properties on the code level, using model checkers and abstract interpreters, e.g. for worst-case execution time analysis [3], [4], which is also regulated by RTCA DO-333, or for replacing[1] at least some of the testing effort [3].

However, the most used verification techniques still remain testing[2] and manual reviews. However, their costs grow disproportionately with the system size [2]. Thus, testing won't be able to deliver exhaustive coverage.

A. Formal Methods

Formal methods tackle the deficits mentioned in section I about testing. Well-known classes of formal methods for being deployed in avionics development are:

- Theorem Proving:
  - Most powerful, most expressive formal method tool.
  - Requires expertise and continuous interaction to use successfully.
- Model Checking:
  - Less expressive than theorem provers.
  - Mostly automated, but still requires expertise to use successfully.
- Abstract Interpretation and Static Analysis:
  - Least expressive, targeted to very specific artifacts.
  - Requires some expertise to discharge false positives.

Theorem provers offer the highest assurance level and have been used for verifying important properties, e.g. the safety and security of the complete kernel of an OS [5]. Formalisms such as Communicating Sequential Processes (CSP) ([6], [7], as used in e.g. [8]), Calculus of Communicating Systems (CCS) [9], π-calculus [10], or FOCUS [11], [12] assist this process considerably, because they support non-determinism

The SpesML project is funded by the German Federal Ministry of Education and Research.

[1] NB, that testing can never demonstrate the absence of issues, but only their presence.

[2] Again, the statement in footnote 1 applies here.

and underspecification, a notion of behavioral refinement, time-sensitive specifications, and hierarchical decomposition. In addition, theorem provers have an advantage (compared to the model checking approach of CSP models in [8]) in particular, when verifying software, since, despite some considerable progress enabled by partial-order approaches [13], model checkers suffer from the explosion of the state space, whereas in theorem proving the complexity of proofs grows only linearly with system complexity [14].

In FOCUS, distributed and interactive systems consist of components exchanging messages through unidirectional channels. The semantics of a component is a (set of) stream processing functions. The most important reason that FOCUS is used in this paper is due to the fact that refinement is fully compositional [12], [15]. This means that after decomposing a system, refining the components separately, and than composing back, the new system will be a correct refinement of the old one. Since the new system cannot exhibit new (potentially unwanted) behaviors per construction, this leads to cost savings for testing and integration efforts.

B. Paper Contribution

The contribution of this paper is updating and continuing our previous projects (German Federal Ministry for Economic Affairs and Climate Action ASSET-2 [16] and German Federal Ministry of Education and Research SPES series [17]–[20]) and works [14], [15], [21]–[24], where the model-based verification of several safety-critical properties is covered. In [14] we used a time-synchronous behavior specification paradigm [15], [25], known to be well-suited for hardware specification and verification [26], where model checking techniques, unlike in software, can usually achieve an excellent exhaustive coverage. Meanwhile, in software applications such as telecommunications, transportation, and business information systems, an event-driven paradigm is much more common to build scalable distributed systems [27]. Hence, in this paper, event-driven automata are introduced for capturing non-determinism and underspecification. Event-driven automata are selected due to the following reasons:

- low-level requirements (LLRs) according to RTCA DO-178C are defined such, that from these the software can be developed without additional input.
- RTCA DO-333 requires models to be created such, that there is enough abstraction regarding several implementation mechanisms.
- It turned out [24], [28], that automata provide an adequate level of abstraction, which simplifies the formal verification of the software's properties in the next step of the verification.
- The verified properties are guaranteed through the refinement, i.e. through the addition of technical details.
- Avionics systems can be regarded as interactive systems, which process input data coming from e.g. sensors or users and generate output data for actuators or to be displayed at the user interface. The processing of the input is driven by events generated by the arrival of e.g. sensor data or according to the user's interaction.

Non-deterministic and underspecified automata are necessary, because we want to develop the system from several perspectives: At the beginning of the development process (see section II-B) a high-level requirement (HLR) can be defined using an underspecified and non-deterministic automaton. The closer we get to the implementation in software code, the less underspecified and the more deterministic the specifying automata's behavior needs to be expressed. This approach enables an iterative refinement of the HLR, which ends at the precise and correct LLR.

In order to cover all mentioned objectives, we extend our previous work by the following key novel contributions:

- Event-driven modeling in SysML [29]
- Event-driven reasoning infrastructure [30]
- Semantical mapping from SysML to a data flow formalism
- Evaluation in a use case from the aerospace domain handling liveness properties

C. Paper Structure

The further structure of this paper is as follows: Section II presents an avionics use case and the methodological approach. Requirements engineering is performed along RTCA DO-178C and RTCA DO-333. Afterwards, our modular development method for cyber-physical systems based on our data flow formalism is described. Event-driven specifications are presented and motivated. Section III presents the domain-specific language used as frontend. A conceptual mapping to the data flow formalism is presented to give semantics to the frontend modeling language. The mapping is implemented by a code generator, which maps from the frontend language to an encoding of the data flow formalism in a theorem prover. Architecture modeling and lower-level requirements are demonstrated on the example of the avionics use case. In section IV the theorem prover is presented. Core encodings of the data flow formalism are introduced. These are used as a reasoning foundation for proving a desired liveness property of the use case, The proof, of which a high-level proof-sketch is presented, reasons over infinite streams. This serves for a verification of the low-level design against abstract requirements, as well as an evaluation for our tool chain. Finally, section V summarizes the results and presents an outlook.

II. USE CASE AND METHODOLOGY

A. Use Case

As exemplary implementation used to evaluate the viability of the methods developed in the joint collaboration between the industrial and academic partners, the development of a representative software functionality of Data Link Uplink Feed (DLUF) (see fig. 1) is selected. In this system the users of a wireless connection need to transfer data packets, which are to be sent with a priority between 1 and 4 (1 denoting the highest priority) to the communication computer $Uplink$. The

messages are received via the I/O element of the computer and placed via a message router in queues (buffers) according to their respective prioritization. These are then read from a $Capacity$ component. The data to be sent are selected from the $Capacity$ component taking into account their priority in such a way, that messages of higher priority have precedence over messages of lower priority. At the same time, the balance of all priority classes has to be achieved, so that higher prioritized messages do not completely rule out the forwarding of messages of lower priority. The data packages, that are viable to be transmitted within a transmission cycle, are finally sent via the component $DataLink$. Figure 1 shows the graphical overview of the architecture.

B. Requirements Engineering

The development of the system, as well as the verification of refinement steps is carried out according to RTCA DO-178C and RTCA DO-333 (see fig. 2) with respect to the safety requirement level Design Assurance Level (DAL) A. The selected system requirements in fig. 3 are used for demonstrating the application of our approach. In particular, for SysReq2 (absence of starvation), an HLR is developed in a formal language (see fig. 7). This HLR component specification has a well-defined interface (input and output ports) and its behavior is given by a formula in SysML v2 expressing the absence of starvation.

We consider this particular requirement (in contrast to the other SysReqs) to be critical enough to be formalized [4]. Liveness properties can only be violated by infinite input streams, so their checking cannot be exhaustively achieved by testing. We thus shall write down the non-starvation requirement as a formal higher-level requirement. Then we will formally verify, that the low-level design fulfills it, which means, that the formal LLR is a correct refinement of the formal HLR. What remains to be performed by manual review is validating that the formalization (of informal SysReqs) was done correctly. This is much less time-intensive, less complex, and less error-prone when compared with an approach without the presented formal verification.

The architecture and LLR (see fig. 1) are defined by giving a white box decomposition of the component, where all formulae (constraint-based specifications, see fig. 7) are replaced by the composition of components, each of which is behaviorally described by (potentially non-deterministic) automata (see fig. 4). Finally, an executable model (e.g. of an atomic component) is the representation of a component by a deterministic automaton (which means the non-deterministic automaton was refined, until it became deterministic). Then the model's execution can be achieved by enabling the user to input a processing order (a user can load default schedulers, or write their own) for (potentially) simultaneously incoming events on this deterministic automaton. Schedulers for executable models, generating source code in a high-level language from the executable model, as well as the generation of executable object code from the source code are not in the scope of this paper.

C. SPES Methodology, FOCUS, and Event-driven Processing

The systematic design process as recommended by RTCA DO-333 is taken into account in essence in ongoing research activities like the SpesML project. The SPES methodology [17] aims at bringing state-of-the-art model-based systems engineering (MBSE) approaches to industry. One of it's main contributions is a well-founded and complete methodology for developing and integrating distributed cyber-physical systems (CPSs).

The SpesML project [20] unites academic and industry partners in an effort to implement and extend the SPES methodology in an industry-approved and -proven tool. The systems modeling language of choice is SysML (v1) [31], because of its wide-spread industry adoption. Our approach is based on the latest version, SysML v2 [29], which is currently under final review. It promises new possibilities, including a textual representation, seamless integration and transformatiom between graphical and textual representation, and a clearer semantics for model elements.

FOCUS [11] serves as a mathematical underpinning of both SPES and our approach. It allows to formally specify and analyze systems [12]. Its key feature is the compositionality of refinements, as explained in section I-A.

Compared to SpesML, we employ the modeling power of event-driven systems. Event-driven modeling of reactions is a more natural fit for typical distributed software systems, especially in time-sensitive environments as they are found in the avionics domain. Our framework models event-driven behavior using state machines [4], that can immediately react to single events like incoming transmissions. After receiving such an input, the system can produce arbitrarily, but finitely many outputs and/or simultaneously change its internal state. Figure 4 shows such an event-driven behavior specification.

Section III will now introduce our revised (compare [14]) SysML v2 modeling language profile and adapted tooling for highly automated formal verification of event-driven systems.

III. A SYSTEM MODELING LANGUAGE PROFILE AND THE MAPPING TO A DATA FLOW SEMANTICS

A. SysML v2 and Code Generator

Working directly with theorem provers requires excessive expertise. We therefore propose a model-driven approach based on a SysML v2 profile. This is similar to SpesML [20], where SysML v1 is used as a frontend to the SPES methodology [17]–[19]. Systems and subsystems are restricted to a universal interface model: Systems are defined using part definitions (blocks in SysML v1). Part definitions communicate only via ports. Behavior of systems is modeled using one of three options: exhibited state definitions, asserted constraints, or composition. Finally, we propose the addition of a refinement relation.

SysML models are transformed automatically to their FOCUS representations, thereby giving a formal semantics to the modeled system. This enables reasoning and deduction over properties such as consistency between high- and low-level specifications. Our previous publication [32] explains

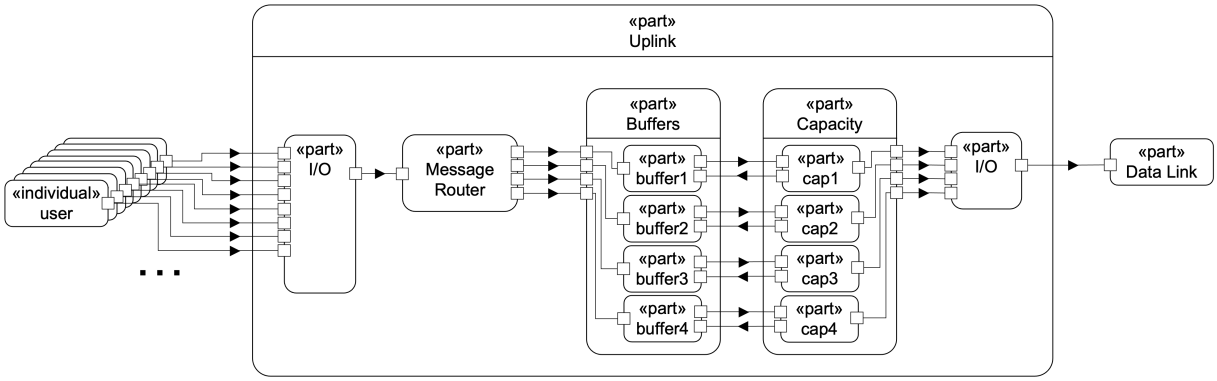

Fig. 1. Graphical representation of the DLUF system in SysML v2 notation. Users send prioritized data packages via an I/O interface to a router. Here, messages are prioritized into four levels and stored in queues (buffers). Messages are then forwarded according the remaining capacity of their particular priority.

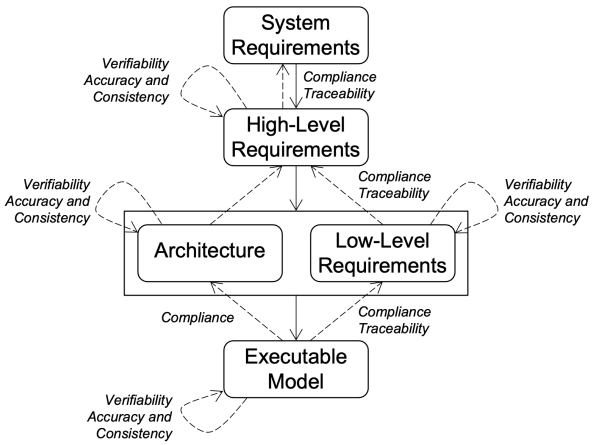

Fig. 2. Systematic design as recommended by RTCA DO-333.

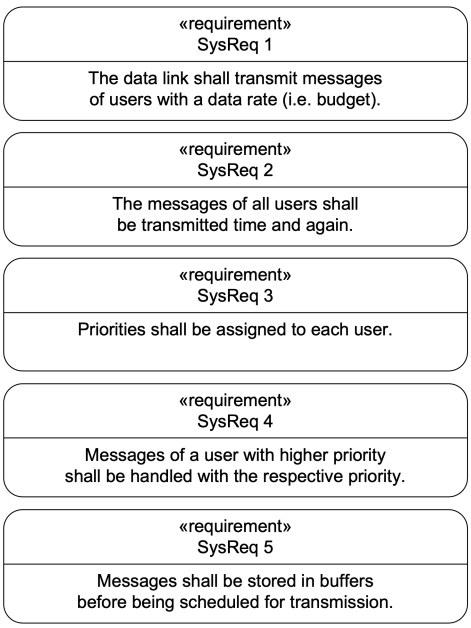

Fig. 3. System requirements.

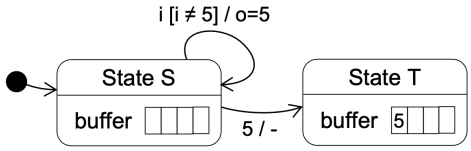

Fig. 4. Example of an event-driven system: The automaton can react to events immediately upon their arrival by producing outputs (top: $o = 5$) and/or changing its state (middle, 5 being inserted into the $buffer$). Actions can be chosen depending on the conditions modeled as guards (top, $i \neq 5$).

this transformation in great detail. The FOCUS representation is embedded into Isabelle [30], an interactive theorem prover originally developed at the University of Cambridge and the Technical University of Munich. Isabelle enables machine-supported and automated proof searches. It can also look for counter examples and execute behavior specifications, i.e. run simulations. Generally speaking, Isabelle allows the generation and verification of machine-based formal proofs, without the possibility of human error.

A novel contribution of this work are event-driven systems. To represent and transform event-driven behavior specifications, we extend our existing [14] FOCUS-compatible state-based systems. Figure 5 gives an overview of key concepts. State space and initial configurations are inherited through an abstract $Automaton$. The newly introduced $EventAutomaton$ enables the representation of event-triggered transitions by grouping them based on their input event's underlying channel. A textual SysML v2 model of an event-driven system was already shown in [32].

We further implemented a user frontend in the form of a Visual Studio Code (VSCode) [33] extension. The extension handles all transformations and prover interactions to automatically derive formal certificates or counter examples. The verification state is indicated using green or red lights, as fig. 6 shows.

B. Safe Architecture and Low-level Requirements

In order to ensure safe LLRs, the consistency with the system requirements needs to be checked. To narrow the gap

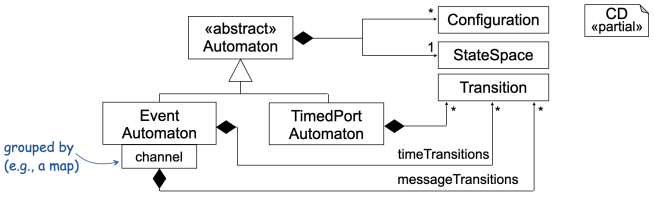

Fig. 5. Adapted intermediary representation of state-based systems compatible with FOCUS compared to [14]: Event-driven automata group transitions based on their input event's underlying channel.

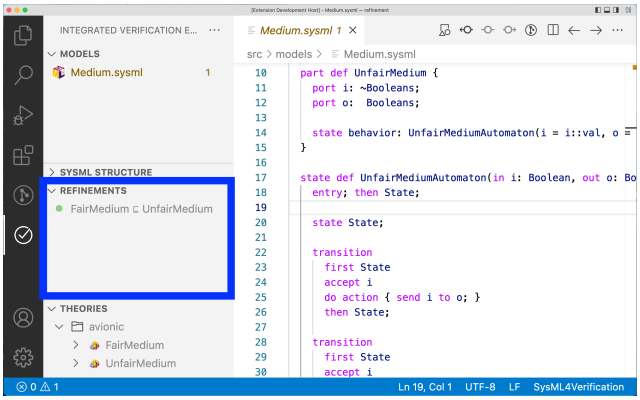

Fig. 6. The user frontend as VSCode extension showing syntax highlighting of a textual SysML v2 model (right), model artifacts (top left), refinement relations including their verification state (blue box in the middle left), as well as underlying automatically generated Isabelle theories (bottom left).

between typically informal system requirements and LLRs, formal HLRs are introduced. Our approach then allows for automated formal verification of consistency between an HLR and its LLR. This is done by proving the correctness of the refinement between the two. Now only the consistency between system requirements and HLRs needs to be checked, e.g. through manual review.

Thus, to ensure non-starvation in our selected system requirement, the absence thereof is encoded in fig. 7. The part definition (keyword $partdef$) in combination with the communication $port$s describes the black box specification, i.e. the interface. There are 4 ports (cardinality 4 in square brackets) in each direction ($\sim$ indicates reversal of flow, the default being outwards). The behavior is restricted (keyword $satisfy$) to non-starving ones using a $requirement$ (a type of constraint, that allows nesting, in order to better structure the desired properties). First, three assumptions ($assumes$) are set. By quantifying over all inputs ($\forall i \in ...$), the length of the observed time frame ($length$) is set to be infinite. Then, it is assured, that each interval t (of type nat) contains some messages ($length() > 0$). Lastly, all input $values$ are assumed to be below the given maximum capacity of the buffers ($maxCap$). The HLR then $requires$, that each output channel holds infinitely many $messages$ (not including time signals).

The HLR is then refined to LLR specifications. In the modelling frontend, this is done using the aforementioned

```
1  part def BufferWithCapacity {
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}. input[i].length() = ∞
8      }
9      assumes 'message in each interval' {
10       ∀i ∈ {1,2,3,4}, ∀t:nat:
11         input[i].atTime(t).length() > 0
12     }
13     assumes 'size below max. capacity' {
14       ∀i ∈ {1,2,3,4}:
15         ∀v ∈ input[i].values():
16           v < maxCap[i]
17     }
18     require 'infinitely many outputs' {
19       ∀i ∈ {1,2,3,4}:
20         output[i].messages().length() = ∞
21     }
22   }
23 }
```

Fig. 7. HLR 2.1 non-starvation formally modeled in SysML v2 textual representation.

refinement relation of section III-A. After automatic transformation to FOCUS and embedding to Isabelle, the refinement can be checked in an automated and tool-supported way. The correct refinement ensures, that all properties (e.g. non-starvation) hold for the refined system (e.g. LLR). This ensures a safe LLR.

IV. THEOREM PROVER ISABELLE

A. Core Encodings

General mathematical FOCUS definitions are specified in the theorem prover Isabelle to serve as a semantical foundation [22] for an automated transformation mapping SysML v2 models to Isabelle theories. This is necessary, because a formal analysis of systems without clear semantics is not possible [34]. This chapter shortly introduces slightly simplified main structures in Isabelle.

The most important data type is the streams domain. Streams are concatenations of messages over some alphabet and describe the history of communication channels in a system. With the keyword $domain$ the stream data type in Isabelle is defined matching the implementation of Haskell lists. A chain-complete partial order (pcpo)[3] with a smallest element [4] [35] is automatically linked to the stream type. Thus, according to [36], [37], a least fixed point exists and infinite streams are approximated by their finite prefixes. This allows

[3]A domain(pcpo) is a partially ordered set in regards to a transitive, reflexive, and anti-symmetric order relation (prefix relation for streams). Stream a is a prefix of stream b, iff each element of a is at the same position in b.

[4]The smallest element for streams is the empty stream, which is a prefix of all streams.

defining semantics for components iteratively processing infinite streams, e.g. event-driven automata. With the established concept of *lazy* evaluation[5] infinite streams are built into the stream domain using constructors.

```
domain 'm stream = cons (head::"'m") (lazy
    rest::"'a stream")
```

Events are either messages or passage of time denoted as tick ($\surd$). Accordingly, an event in Isabelle is defined:

```
datatype 'm event = Event 'm | √
```

With streams and events implemented in Isabelle, it is immediately possible to work with event streams.
A component of a distributed system often communicates with a multitude of other components over multiple input and output channels. The interaction between components is possible with composition, where the interfaces of different components are connected according to their channel names. Because a tuple representation for the communication history of complete component interfaces obstructs a general composition operator formalization, another approach is needed. The concept of *bundling streams* implemented in Isabelle as functions mapping channels to communication histories is suitable for composing arbitrary components. Channels allow only messages specified by the component interface to flow, e.g. boolean messages on one channel or integers on another. Thus, a function mapping channel names to arbitrary streams is not always a stream bundle. All messages sent in a channel history must be allowed messages of that channel:

```
definition wellformed ::
  "('cs ⇒ M stream) ⇒ bool" where
"wellformed f = ∀channel. messagesOf(f channel
    ) ⊆ allowedOn channel"
```

With the *pcpodef* keyword, the type of stream bundles is defined as all wellformed functions. Additionally, the complete partial order is given automatically.

```
pcpodef 'channel bundle("(_^Ω")
  = "{f::('channel ⇒ M stream). wellformed f}"
```

The definition also lifts the prefix order over streams pointwise to stream bundles. The composition operator is not only capable of sequential and parallel, but also of feedback compositions[6], and is implemented in [22].

A deterministic component processes input elements via its input channels and maps them to outputs on output channels. Using bundles as input and output interface abstractions, each continuous function over bundles describes a deterministic component and is called stream processing functions (SPFs) in FOCUS.

```
type_synonym ('I,'O) spf = "'I^Ω →' O^Ω"
```

To be able to analyze systems even early on in the development, components with different behavior implementation possibilities must be part of the semantics domain. A set of SPFs, where each SPF is one implementation possibility, is called stream processing specification (SPS). Since components represented by multiple SPFs have different possible behavior, they are underspecified.

```
type_synonym ('I,'O) sps = "('I,'O) spf set"
```

An event-driven component processes each input event individually. For the behavior description the structure of an event-driven automaton is added. The step wise evaluation is described by transitions mapping an input event to possible next states and output events with regards to the current state. Since an event-driven component starts in a state and can produce initial output, a set of possible initial configurations is another essential part of the automaton. The transition function as well as initial configurations are able to represent different possibilities to capture a component's underspecification. The state space and input and output alphabet are parameters for the *Automaton* type in Isabelle:

```
record ('state,'message,'outchannel)
Automaton =
 transitions::"'state ⇒ 'message ⇒ (('state ×
     'outchannel^Ω) set)"
 configurations::"('state × 'outchannel^Ω) set"
```

A semantical mapping, mapping event-driven automata to SPS was implemented according to [39] and fully embeds the automaton type in the existing FOCUS framework in Isabelle.

The core encodings are extended by additional functions over bundles, SPFs, and SPSs. Functions needed to formulate theorems proving this paper's system requirements are:

- $timeSlice\ t\ s$, obtaining all ordered messages starting at time t until time $t+1$
- $values\ s$, obtaining all messages of the communication history s
- $\#_P\ s$, returning the number of occurrences of messages fulfilling predicate P in s[7]
- $\bigotimes$, the composition operator lifted to SPS'.

Implementation details, other functions, and general theorems are introduced in [22].

B. Property Verification

In order to ensure, that even low priority messages are transmitted again and again, we formally verify a non-starvation property for the system. It confirms, that always receiving messages of a specific priority implies transmitting messages of that priority infinitely often. The proof of such a property over a composed system is, due to the unique compositionality property of FOCUS, reduced to proving properties for its subcomponents. In other words, the correctness of a composed system is the direct result of the correctness of its components. Complex formal verification over systems can be divided and conquered in many smaller steps with automation possibilities. In this case, the non-starvation property is traced back to a non-starvation property of the buffer and the capacity components.

[5]Evaluation strategy to only evaluate expressions, when their value is required to progress [38].

[6]A composition with feedback is determined by a fixed-point calculation.

[7]For a tautology P, the function $\#_P$ returns the length of the stream.

The following properties are formulated and proven in Isabelle for the buffer components:

1. If each input time slice contains packages, so does each output time slice.

```
theorem assumes "buffer_i ∈ BufferSPS_i"
and "∀time. timeSlice time packets_i ≠ ϵ"
and "output_i = buffer_i(packets_i,confirms_i)"
shows "∀time. timeSlice time output_i ≠ ϵ"
```

2. All output packages were received input packages.

```
theorem assumes "buffer_i ∈ BufferSPS_i"
and "output_i = buffer_i(packets_i,confirms_i)"
shows "values output ⊆ values input"
```

Together with the following property for the capacity components

If each input time slice contains packages and the package sizes are smaller or equal to the maximal capacity, infinitely many messages are transmitted.

```
theorem assumes "capacity_i ∈ CapacitySPS_i"
and "∀time. timeSlice time packets_i ≠ ϵ"
and "∀message ∈ values packets_i. size message
    ≤ maxCapacity_i"
and "(output_i,confirms_i) = capacity_i(packets_i)"
shows "#_packets (output_i) = ∞"
```

they immediately imply the non-starvation property for the composed system:

If each input time slice of a priority contains packages and the package sizes are smaller or equal to the maximum capacity of that priority, infinitely many packages of that priority are transmitted.

```
theorem
assumes "transmit_i ∈ (BufferSPS_i ⨂ Capacity_i)"
and "∀time. timeSlice time packets_i ≠ ϵ"
and "∀message ∈ values packets_i. size message
    ≤ maxCappacity_i"
and "(output_i) = transmit_i(packets_i)"
shows "#_packets (output_i) = ∞"
```

The non-starvation property holds for each priority. The continuous transmission of packages is ensured.

V. Conclusion

The methodology demonstrated in this paper consists of a model-based verification framework enabling event-driven system specifications and reasoning. It enables a verified design and a correct refinement of safety-critical systems. The designer can either directly specify the system using a logic language such as Isabelle, or using an architecture description language such as SysML as a user-friendly way for describing the interface, behavior, and interaction between components. The system model and any desired properties can then be translated to equivalent specifications in a theorem prover. FOCUS as semantical foundation stands out due to the compositionality of refinements. This means that after decomposing the system, refining the subcomponents separately e.g. until an implementation, and then composing back, the new system is a correct refinement of the old one and no further integration tests are needed. This can lead to the replacement of a considerable amount of manual tests and reviews. This also helps with requirements demanding properties being true always or never, which generally cannot be fully verified by testing. Note however, that certain sets of tests and reviews can only be supplemented by this approach, but not completely replaced. In particular, the following checks have to be performed manually:

- The formalization of the requirements is correct (i.e. the correct transformation from informal to formal models).
- The methodology is justified and appropriate.
- Requirements and software architecture are compatible with the target computer (unless the target environment is formally modeled).
- The requirements are complete.
- There is no unidentified dead or disabled code.

In general, we observe an increasing maturity and feasibility in the application of formal methods in safety-critical systems, as it is possible by following the RTCA DO-333 standard, which can help to replace or complement many tests. Note, that the formal specification might create some additional effort, when considering the overall benefits over testing. However, they usually overcompensate later significantly, since technical flaws at the beginning may result in highly expensive corrections of deficits identified later in the development process, and the later the errors are corrected, the more costly they are to correct. Finally, analyzing the design at early stages in the development process can help to optimize the architecture layout of avionics systems, leading to weight reduction, less energy consumption, and less CO_2 emissions.

References

[1] B. Annighoefer, M. Halle, A. Schweiger, M. Reich, C. Watkins, S. van der Leest, S. Harwarth, and P. Deiber, "Challenges and ways forward for avionics platforms and their development in 2019," in *38th Digital Avionics System Conference (DASC)*, 2019.

[2] A. Brahmi, D. Delmas, M. H. Essoussi, F. Randimbivololona, A. Atki, and T. Marie, "Formalise to automate: deployment of a safe and cost-efficient process for avionics software," in *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, (Toulouse, France), Jan. 2018.

[3] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate, "Testing or formal verification: Do-178c alternatives and industrial experience," *IEEE Software*, vol. 30, no. 3, pp. 50–57, 2013.

[4] U. Schopp, A. Schweiger, M. Reich, T. Chuprina, L. Lucio, and H. Bruning, "Requirements-based code model checking," in *2020 IEEE Workshop on Formal Requirements (FORMREQ)*, (Los Alamitos, CA, USA), pp. 21–27, IEEE Computer Society, sep 2020.

[5] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser, "Comprehensive formal verification of an os microkernel," *ACM Trans. Comput. Syst.*, vol. 32, Feb. 2014.

[6] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[7] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall International, 1985.

[8] T. Murray and G. Lowe, "On refinement-closed security properties and nondeterministic compositions," *Electr. Notes Theor. Comput. Sci.*, vol. 250, pp. 49–68, 09 2009.

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

[10] J. Parrow, "An introduction to the pi-calculus," 2001.

[11] M. Broy and K. Stølen, *Specification and development of interactive systems: Focus on streams, interfaces, and Refinement*. New York: Springer, 2001.

[12] M. Broy and B. Rumpe, "Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung.," *Informatik Spektrum*, vol. 30, no. 1, pp. 3–18, 2007.

[13] J. Esparza and K. Heljanko, "Unfoldings - a partial-order approach to model checking," in *Monographs in Theoretical Computer Science. An EATCS Series*, 2008.

[14] H. Kausch, M. Pfeiffer, D. Raco, and B. Rumpe, "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* (S. Götz, L. Linsbauer, I. Schaefer, and A. Wortmann, eds.), vol. 2814, CEUR, February 2021.

[15] H. Kausch, M. Pfeiffer, D. Raco, and B. Rumpe, "Montibelle - toolbox for a model-based development and verification of distributed critical systems for compliance with functional safety," in *AIAA Scitech 2020 Forum*, p. 0671, January 2020.

[16] M. Reich, A. Schweiger, J. Lorenz, and U. Margull, "Experience report on reuse in avionics," in *IBS Workshop Micro Air Vehicle Technologie – Konzepte und Anwendungen 2019* (W. Hardt, ed.), vol. 9, pp. 28–35, TUDpress THELEM Universitätsverlag GmbH und Co. KG, 2020.

[17] K. Pohl, H. Hönninger, R. Achatz, and M. Broy, eds., *Model-Based Engineering of Embedded Systems*. Heidelberg: Springer Berlin, 2012.

[18] K. Pohl, H. Hönninger, H. Daembkes, and M. Broy, eds., *Advanced Model-Based Engineering of Embedded Systems*. Basel: Springer Cham, 2016.

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

[20] Official SpesML Project Site `https://spesml.github.io`.

[21] J. Ringert and B. Rumpe, "A little synopsis on streams, stream processing functions, and state-based stream processing," *Int. J. Software and Informatics*, vol. 5, pp. 29–53, 01 2011.

[22] J. C. Bürger, H. Kausch, D. Raco, J. O. Ringert, B. Rumpe, S. Stüber, and M. Wiartalla, "Towards an Isabelle Theory for distributed, interactive systems - the untimed case," Tech. Rep. AIB-2020-02, RWTH Aachen University, March 2020.

[23] H. Kausch, M. Pfeiffer, D. Raco, and B. Rumpe, "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* (R. Hebig and R. Heinrich, eds.), vol. 2581, CEUR Workshop Proceedings, February 2020.

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

[25] R. Grosu and B. Rumpe, "Concurrent timed port automata," *arXiv preprint arXiv:1411.6027*, 2014.

[26] J. He and K. J. Turner, *Specification and Verification of Synchronous Hardware using LOTOS*, pp. 295–312. Boston, MA: Springer US, 1999.

[27] S. Kounev, C. Rathfelder, and B. Klatt, "Modeling of event-based communication in component-based architectures: State-of-the-art and future directions," vol. 295, 03 2012.

[28] U. Schöpp, A. Schweiger, M. Reich, T. Chuprina, L. Lúcio, and H. Brüning, "Requirements-based code model checking," in *2020 IEEE Workshop on Formal Requirements (FORMREQ)*, pp. 21–27, 2020.

[29] SysML v2 Submission Team, "OMG Systems Modeling Language (SysML)," tech. rep., Aug. 2021.

[30] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A proof assistant for Higher-Order Logic*, vol. 2283 of *Lecture notes in artificial intelligence*. Berlin [etc.]: Springer, 2002.

[31] Object Management Group, "SysML Specification Version 1.0 (2006-05-03)," August 2006. http://www.omg.org/docs/ptc/06-05-04.pdf.

[32] H. Kausch, J. Michael, M. Pfeiffer, D. Raco, B. Rumpe, and A. Schweiger, "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), November 2021.

[33] Microsoft `https://code.visualstudio.com/`.

[34] D. Harel and B. Rumpe, "Meaningful Modeling: What's the Semantics of "Semantics"?," *Computer*, vol. 37, no. 10, pp. 64–72, 2004.

[35] B. C. Huffman, *HOLCF '11: A definitional domain theory for verifying functional programs*. [Portland, Or.]: Portland State University, 2012.

[36] R. P. Agarwal, M. Meehan, and D. O'regan, *Fixed point theory and applications*, vol. 141. Cambridge university press, 2001.

[37] A. Granas and J. Dugundji, *Fixed point theory*, vol. 14. Springer, 2003.

[38] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM Computing Surveys (CSUR)*, vol. 21, no. 3, pp. 359–411, 1989.

[39] B. Rumpe, *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. Doktorarbeit, Technische Universität München, 1996.