Automated Generation of Requirements for the Highly Fault-Tolerant System Behaviour of a Distributed and Integrated Avionics Platform

Robert Wipperfürth, Thorben Hoffmann, Christoph Kurz, Tim Belschner, Reinhard Reichel

Automated Generation of Requirements for the Highly Fault-Tolerant System Behaviour of a Distributed and Integrated Avionics Platform

Robert Wipperfürth*†, Thorben Hoffmann*, Christoph Kurz*, Tim Belschner* and Reinhard Reichel*
* *Institute of Aircraft Systems*
University of Stuttgart
Stuttgart, Germany
† robert.wipperfuerth@ils.uni-stuttgart.de

***Abstract*—Fully autonomous Unmanned Aerial Vehicles, Remotely Piloted Aircraft, Air Taxis, as well as advanced CS-23 aircraft require numerous complex and safety-critical system functions, such as vehicle management and utility functions, automatic take-off and landing or flight control. The development and qualification of the related avionics systems are characterised by a very high effort. The Institute of Aircraft Systems at the University of Stuttgart, in close cooperation with Aviotech GmbH, aims at a highly automated development and verification process for such fault-tolerant avionics systems to significantly reduce development effort, time, and risk and thus costs. For this reason, the Flexible Avionics Platform was developed. It enables the implementation of integrated fly-by-wire platform instances and is characterised by the following key aspects. (1) A platform-based development approach featuring an integrated, distributed, and highly redundant avionics architecture. (2) The platform management, a high-level abstraction layer providing a full abstraction towards integrated applications regarding the distribution, fault-tolerance, and redundancy of a fly-by-wire platform instance including redundant peripherals. (3) The AAA process, a comprehensive automation process for the highly automated generation of development and qualification artefacts, such as an instance of the Platform Management, the corresponding specification at the system and software level, and related test cases and test scripts. This paper presents the basics for the automated requirements generation at the system level with a focus on the specification of the highly fault-tolerant system behaviour of fly-by-wire platform instances based on the Flexible Avionics Platform.**

***Index Terms*—Flexible Avionics Platform, AAA process, platform-based development, model-driven development, automatic requirements instantiation, requirements reuse**

I. Introduction

A. Motivation

Future aircraft require numerous complex and safety-critical system control functions. This is especially true for fully autonomous Unmanned Aerial Vehicles (UAVs) and Remotely Piloted Aircrafts (RPAs) that are certified and operated in non-segregated airspace. Such aircraft must be able to react in a semi- or fully-automated manner to all critical situations as, for instance, the loss of the command and control data link or the loss of an engine. Furthermore, standard operational functions like take-off and landing or the entire vehicle management and utility functions must be automated as well.

Accordingly, the implementation of the related avionics systems is characterised by a very high effort. An implementation based on avionics platforms with highly integrated architectures is a first step to reduce system costs. While this approach enables the integration of a high number of systems within a single instance of such an avionics platform, adequate safety, up to failure rates of $10^{-9}/h$, has to be ensured. To enable an affordable usage of such integrated avionics platforms, especially within the CS-23 domain, a new approach is required that reduces development and qualification effort significantly while maintaining a high safety level.

B. State of the Art

In the avionics domain, Integrated Modular Avionics (IMA) represents the state of the art regarding platform-based development approaches as well as integrated and distributed architectures [1], [2], [3]. The hardware is based on standardised modules connected via an Avionics Data Communication Network (AFDX®). Concerning the software architecture, the hardware and communication are abstracted towards integrated applications (see Fig. 1). This includes inter-partition communication, even if the partitions are allocated to different hardware modules. There are comprehensive automation approaches for the automated instantiation of these IMA abstraction layers [4], [5], [6], [7].

Due to these features, IMA represents a significant advance in the realisation of avionics systems, their development, and their qualification. However, despite all process automation approaches, the development and qualification of avionics systems based on IMA still require a high effort. The reason for this is the low abstraction level provided by IMA. IMA does not provide an abstraction of system management aspects such as the management of redundant sensors, redundant actuators, and the failure and redundancy management of distributed modules ensuring consistency in a distributed architecture. Thus, this extremely complex part of the system management, especially with highly redundant avionics systems, has to be

This research was funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) within the LuFo V-2 and LuFo V-3 program.

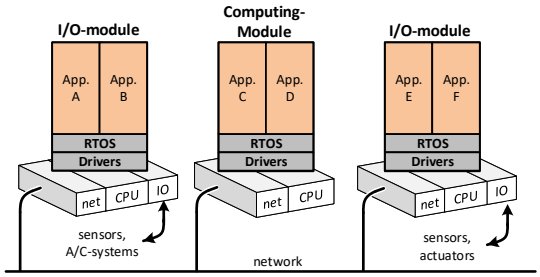

Fig. 1. Integrated Modular Avionics architecture.

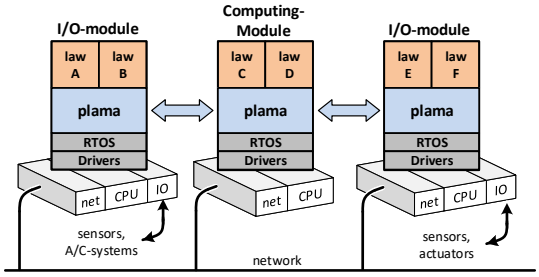

Fig. 2. Flexible Avionics Platform architecture.

implemented as part of the integrated applications. Consequently, the most challenging tasks of the system management are not part of the IMA development automation. This is the key difference to our approach of the Flexible Avionics Platform with its associated process automation.

C. The Flexible Avionics Platform [8]

The Flexible Avionics Platform (apf) is a platform-based development approach in the sense of Di Natale et al. [9] and features an integrated, distributed, and highly redundant architecture. It consists of standardised hardware components as well as a library of generic software components, i.e. a Real Time Operating System (RTOS), drivers, and small software bricks called basic services. Each specific avionics system is generated as an instance of this Flexible Avionics Platform.

The apf is characterised by a high-level abstraction layer realised as a middleware, the so-called Platform Management (plama) (see Fig. 2). It covers the entire system management for all distributed and redundant modules (internal and external) as well as redundant peripherals, for example, redundant sensors and redundant actuators. Thus, it manages tasks such as fault-tolerant intra-module and inter-module communication, communication to other systems, the redundancy of all modules and peripherals, and reconfigurations in the event of failure. Furthermore, it controls the system-wide operation phases such as *normal operation*, *pre-flight built-in test*, and *interactive system operation* during maintenance. These system management tasks are handled by plama in a distributed manner and are executed on all modules of a Flexible Avionics Platform Instance (apfi). Each plama instance is implemented as a composition of specialised basic services.

Due to the high degree of abstraction provided by plama, integrated applications are executed in a failure-free virtual simplex environment and are not required to perform any system management tasks. Hence, applications are reduced to their cybernetic control law.

D. The AAA Process

The apf is complemented by a comprehensive automation process for the development and qualification of an apfi, the AAA process. It consists of the following subprocesses:

- Axx subprocess – Automated design and parameter instantiation [10]: Based on a high-level system specification a high-level system design model is implemented manually. It is expressed with a domain-specific modelling language developed by the Institute of Aircraft Systems (ILS). This model describes the basic hardware structure, connected sensors and actuators, interfaces to other systems, a placeholder for the applications, the degree of redundancy, the basic reconfiguration strategy, and the scheduling of the apfi-wide operation phases.
  In a first automation step, the implemented system design model is refined by synthesis rules into a software architecture model containing the key software components and their coupling. In a subsequent automation step, the software architecture model is further refined into a model of the software components' Parameter Data Items (PDIs). The PDI model defines the selection of suitable basic service, their functional specialisation, and their data and control coupling. In addition, it defines the configuration data for all drivers and the RTOS. As a last step, this model is automatically transformed into source code. This PDIs source code together with the source code of the selected basic services, drivers, and the RTOS as well as the hardware modules form an apfi.
- xAx subprocess – Automated document generation [11]: The approach of the automatic generation of requirements is based on requirement classes describing characteristic behaviour that can be realised with the apf. Requirement classes are modelled once in terms of characteristic patterns, defining aspects such as their instantiation condition, representation, and the structure of possible instances. The corresponding models are automatically transferred into synthesis rules. Executing these synthesis rules, the xAx tool suite automatically analyses the apfi design models, generated in the Axx subprocess, with regard to the characteristic patterns of each requirement class. The match of a single or a group of patterns leads to the automatic generation of a requirement instance, i.e. the specialisation of the related requirement class with the apfi specific information of this match.
  In this way, all apfi requirements are generated automatically for system-level, software high-level and software low-level. In general, a requirement instance is expressed in the form of a natural language representation and a formal representation. The natural language representations are human-readable descriptions of the requirements and include information on traceability and versioning.

They conform to the relevant aeronautical standards ARP4754A and DO-178C and represent a large part of the entire apfi specification documents that are used for an apfi certification process. The formal representations feature corresponding and consistent models of the apfi behaviour and are further processed by the xxA subprocess.

- xxA subprocess – Automated generation of verification artefacts [12], [13]: Based on the formal representations the xxA tool suite automatically generates associated test cases and test procedures respectively test scripts using a dedicated test oracle. During the integration and verification processes for an apfi, these test scripts can be automatically executed on an apfi-in-the-loop testrig.

E. Automated Requirements Generation for Plama's High-Level Management Layer

A first approach for the automated generation of requirements focused on plama's communication layer (low-level management) which manages the fault-tolerant intra-module and inter-module communication as well as the monitoring and data fusion [11].

This paper presents the enhancement of the approach for the system specification of the high-level management layer of plama, i.e. the layer managing reconfigurations and the operation moding of an apfi. The strategy for an automated generation of requirements is based on the following considerations. The automatically generated instances of the requirement classes have to enable a meaningful apfi specification that can be used for validation processes and that is comparable to a manual specification. In addition, the set of existing requirement classes has to cover the entire usage domain of the apf. Especially at the system level, these requirement classes should describe the main characteristics. Furthermore, the effort considering the manual implementation of requirement classes must be justifiable.

For this reason, the behaviour of an apfi at the system level must be characterised by a limited number of generic requirement classes valid for all apfis. The central question resulting from this is how these requirement classes are to be defined. This is the focus of our article.

F. Paper Structure

First, we set the context by showing central architectural and operational aspects of an apfi in section II. While section III analyses what must be specified for an apfi at the system level, section IV elaborates on how the system-level behaviour is specified using requirement classes. Finally, section V summarises the key results and provides an outlook on our ongoing and future research.

II. Flexible Avionics Platform

The section is intended to clarify the framework of the apf. For this purpose, the architecture of the apf is presented and important operation principles are explained.

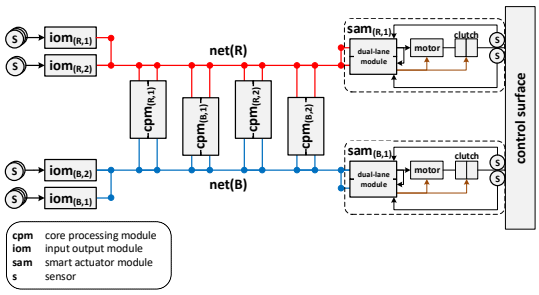

Fig. 3. Hardware structure of an exemplary instance of the apf.

A. Architecture

Figure 3 shows the hardware structure of an exemplary instance of the apf consisting of input output modules (ioms), core processing modules (cpms), smart actuator modules (sams), as well as the networks (net). An iom provides necessary input/output interfaces to connect system-specific peripheral units such as sensors, actuators, human-machine interfaces, or other systems. Measured Signals are preprocessed and transmitted to the cpms. A cpm is a dual-lane module. Each lane operates cyclically and all tasks are executed in parallel and frame synchronously with the other lane. A cpm features a fail/passive behaviour. This is realised by the cross-comparison mechanisms of plama, which ensure the required consensus [14] between the lanes. Cpms have two main tasks. First, they plan and execute the central management decisions within an apfi. Second, they execute the laws of integrated applications, i.e. integrated system function laws. Control commands for the operation of the control surfaces of an aircraft are transmitted to the sams. A sam operates and monitors a control surface. It features a dual-lane architecture including the power electronics for an electric actuator with an optional clutch. All modules communicate via two independent networks, which can provide further internal redundancy. As shown, each module is logically assigned to one network side red $S_{(R)}$ or blue $S_{(B)}$. For example, $cpm_{(R,1)}$ with module ID 1 is assigned to network side $S_{(R)}$.

B. Apfi Operation

The apfi including connected peripherals and all integrated system functions must operate correctly in numerous operation phases as well as numerous apfi configurations. Here, an apfi configuration describes the state of an apfi regarding the dynamic configuration of the resources taking part in the apfi operation. It is not to be confused with the PDI generated by the Axx tool suite. If the apfi operation is affected by a failure, apfi reconfiguration measures are required. Likewise, the operation phase must be adjusted if the apfi operation conditions change. The decision making regarding such actions is designed strictly hierarchically [15], [16]. The corresponding decision hierarchy is as follows:

1) Determination of the membership (membership determination).
2) Allocation of multi-application (mapp allocation).

3) Allocation of the master-slave engagement (master-slave allocation).
4) Scheduling of the operation phases (operation moding).

In general all apfi modules, as well as the connected peripheral units, are considered within this hierarchy. However, the following sections focus on the apfi core which comprises all cpms of the distributed apfi.

1) Membership Determination: The basis for consistent distributed decisions of the cpms is the so-called membership. It ensures that only non-faulty cpms participate in the apfi operation.

The membership determination is a two-step process. First, each cpm determines the membership of its own module locally (module membership). Based on this, each cpm determines the membership of the other cpms (inter-cpm membership). For the inter-cpm membership consensus between all correct cpms is required.

2) Mapp Allocation: The apfi features a dynamic in-flight reallocation of integrated system function laws (sfls) between cpms thus allowing for an efficient use of the cpm hardware. For this, several sfl of the same criticality level are grouped to a multi-application (mapp). Mapps are prioritised according to their criticality. For example, $mapp_{(1)}$ contains the absolute safety-critical system functions *flight control*, *braking*, and *steering* while $mapp_{(2)}$, with the lower priority, comprises functions such as *flight guidance* or *flight management*. Although each cpm has loaded the software for all mapps, it executes only a single mapp at a time.

The assignment of which cpm executes which mapp is called mapp allocation. Mapps are allocated only to such cpms for which membership is given. The mapp allocation is based on apf generic mapp allocation rules. If the rules are met, the mapp allocation is correct. If the rules are violated, the corresponding incorrect mapp allocation is changed to a new, correct mapp allocation. This mapp reconfiguration may take place in flight. For the mapp allocation, consensus is required between all cpms with a given membership.

3) Master-Slave Allocation: If the same mapp is allocated to multiple cpms, i.e. if there are multiple replicas of a mapp, an active-hot-standby replication strategy is applied. Only data provided by the active mapp-replica, i.e. the mapp engaged as master, is processed by other mapps, ioms, or sams. The data provided by the hot-standby mapp-replica, i.e. the mapp engaged as slave, is transmitted via the network but is ignored by all recipients[1]. If there are further mapp-replicas, these are operated as cold-standby replicas. In our article, this so-called shadow engagement is not considered any further.

The selection of which mapp-replica is to be operated as master or slave is called master-slave allocation. As with the mapp allocation, the master-slave allocation is also based on an apf generic ruleset. These rules ensure that the mapp-replica with the best performance execution level[2], regarding the implemented sfls of a mapp, is selected as the master. For the master-slave allocation, consensus between all replicas of a mapp is required.

4) Operation Moding: During a mission, the system of integrated systems, i.e. the apfi with all its integrated system functions including connected peripheral units, undergoes various operation phases. For each operation phase, the required behaviour can change. Examples are the normal operation during flight, the execution of all test steps of a pre-flight built-in test before dispatch, or the interactive activation of built-in tests during maintenance.

Each operation phase is associated with an apfi specific set of hierarchically structured operation modes that can be specified in the high-level system design model for the Axx subprocess. These operation modes define the behaviour of the entire system of integrated systems. The scheduling of the operation modes, which mainly depends on the apfi operation conditions, is performed centrally by the master replica of $mapp_{(1)}$ ($mapp_{(1,ma)}$). Other replicas of $mapp_{(1)}$ simply adopt the current operation modes determined by $mapp_{(1,ma)}$ which ensures consensus.

5) Examplary Apfi Reconfiguration: To provide an insight into the apfi operation, a failure induced mapp and master-slave reconfiguration is presented exemplarily. The initial state is shown in Fig. 4(a). The module membership of each cpm, as well as the inter-cpm membership on all other cpms, is set to ON. Thus, membership is given for all cpms. The shown mapp and master-slave allocation is correct.

Starting from this state, there is a failure in one lane of $cpm_{(R,1)}$. Due to the dual-lane architecture, this failure is detected by the other correct lane of $cpm_{(R,1)}$, which then passivates the cpm. To do this, the correct lane changes its module membership to OFF and stops the transmission of messages over the network.

The passivation of $cpm_{(R,1)}$ which executed $mapp_{(1,ma)}$ affects the other cpms of the apfi. They cannot communicate with $cpm_{(R,1)}$ any more. Hence, the remaining correct cpms consistently set the inter-cpm membership about $cpm_{(R,1)}$ to OFF. However, the mapp and master-slave allocation of the cpms with given membership violate the allocation rules:

1) Although the priority of $mapp_{(1)}$ is higher than the priority of $mapp_{(2)}$, it is allocated to only one cpm.
2) There is no master replica of $mapp_{(1)}$.

Consequently, the mapp allocation and the master-slave allocation have to be reconfigured. In a first step, $cpm_{(B,1)}$ reconfigures from $mapp_{(1,sl)}$ to $mapp_{(1,ma)}$. In a second step, $cpm_{(R,2)}$ reconfigures from $mapp_{(2,sl)}$ to $mapp_{(1,sl)}$. The state illustrated in Fig. 4(b) represents the resulting, correct allocation. Since the apfi operation condition did not change, the new $mapp_{(1,ma)}$ maintains the operation modes adopted from the previous master-replica.

III. Overview of the Specification of an Apfi at System Level

This section aims to illustrate what must be specified at the system level. For this purpose, the classification of the

[1]If the master-slave allocation changes, this enables a transient-free usage of data from the new master by the recipients.

[2]The performance level accounts for the state of sensors and actuators being part of the execution of the system functions.

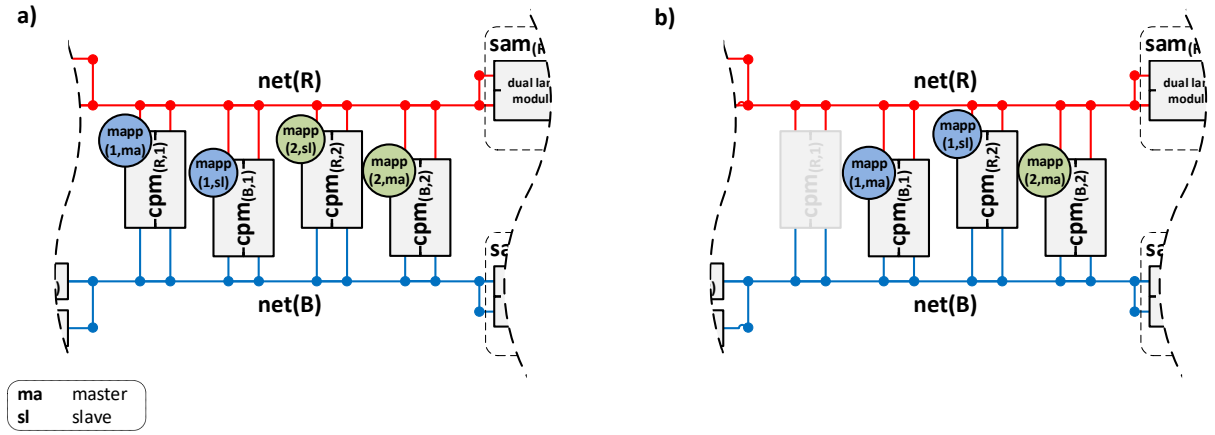

Fig. 4. Mapp and master-slave reconfiguration due to a failure in $cpm_{(R,1)}$.

specification is explained and details on each specification domain are provided.

A. Basics

Most of the time, the apfi operates in a steady-state. A steady-state is characterised by a steady apfi configuration and a steady operation phase. Hence, all allocation rules are met and the required behaviour of the system of integrated systems is stationary. However, failures or changes in the aircraft operation condition can lead to the transition of one steady-state to another. Based on this, the apfi specification at the system level is divided into the following specification domains:

1) The specification of the steady-state operation.
2) The specification of transitions between steady-states.

B. Specification of the Steady-State Operation

The requirements of this domain specify the correct operation of an apfi for all steady-states. A correct operation of a steady-state is characterised by

- the data flow,
- the operation features.

Considering the data flow, data required for the execution of a system function must be correctly transferred from the input of the apfi (e.g. sensors) to the application programming interface of the sfls integrated on each cpm. Furthermore, data must be transferred between sfls and from an sfl to the output of the apfi (e.g. actuators).

Regarding the operation, key features such as the cyclic operation, synchronicity, and consensus, have to be ensured.

C. Specification of Transitions between Steady-States

This specification domain defines the system-level transitions from one steady-state to a new steady-state, due to failures or changes in the aircraft operation condition. Thus, the requirements of this domain specify the reconfigurations and operation moding of an apfi at the system level. Based on the decision hierarchy presented in section II-B, the specification domain is further divided into subdomains.

- The intra-module subdomain specifies the reconfiguration behaviour of single modules.
- The inter-module subdomain specifies the reconfiguration of sets of modules. This includes reconfigurations of the inter-cpm membership, the mapp allocation, or the master-slave allocation.
- The peripheral subdomain specifies reconfigurations affecting the operation of sensors and actuators as part of the execution of system functions.
- The operation moding subdomain specifies the scheduling of operation modes for the system of integrated systems.

For each of these specification subdomains, there is a dedicated set of requirement classes, which together cover all possible aspects of that subdomain. This facilitates a focused and expressive specification of the related behaviour. The sum of all requirement classes covers the overall reconfiguration and operation moding behaviour for all possible apfis. In the following, exemplary requirement classes for an apfi reconfiguration are presented.

IV. Requirement Classes at System Level

This section exemplarily describes how the requirement classes of apfi reconfigurations at the system level are defined. For this, the basic reconfiguration principle of the apf is shown. This is followed by exemplary requirement classes covering different aspects of a reconfiguration.

A. Basic Reconfiguration Principle

Figure 5 illustrates the basic reconfiguration principle valid for each apfi. An apfi features a large number of possible failure events such as a power interruption, CPU failures, or sensor failures. If a failure event ev_i is monitored by a dedicated failure detection mechanism, the corresponding particular indication $z_{I,i}$ is set. Particular indications are then fused to so-called categorical indications $z_{CI,x}$ which are confirmed in time. While the events and the particular indications are mostly apfi specific, the set of possible categorical indications is generic for the apf. Because of the large usage

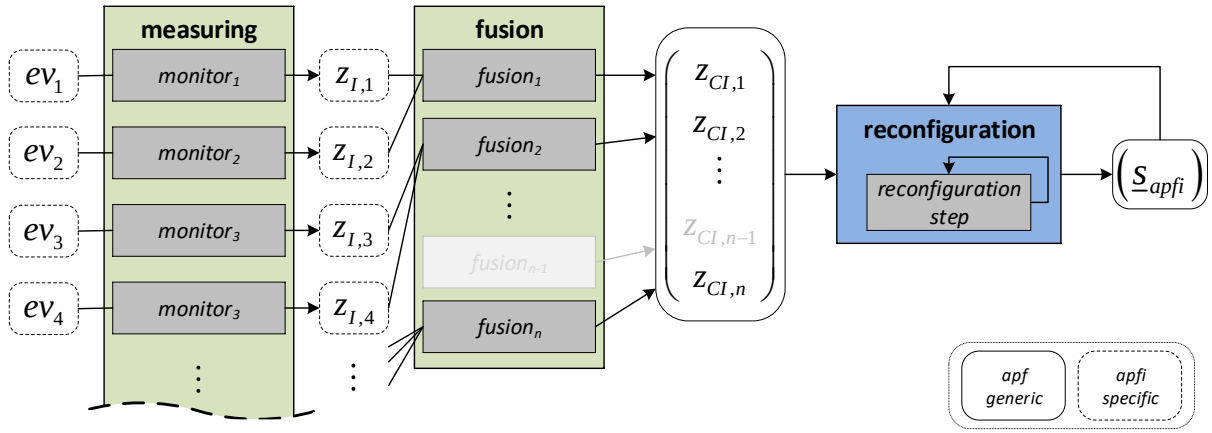

Fig. 5. Illustration of the basic reconfiguration principle of the apf.

domain of the apf, in general, only a subset of all possible categorical indications is needed for a specific apfi[3].

Necessary reconfiguration steps for the current apfi configuration $\underline{s}_{apfi}$ are planned exclusively based on these categorical indications. The steps are executed over a short period until the overall reconfiguration is completed, i.e. until a new steady-state of the apfi configuration is re-established. As each apfi reconfiguration is associated with a categorical indication, the set of all possible reconfigurations is generic for the apf as well. In the following, the requirement class for the fusion of particular indications is introduced, which enables the specification of the categorical indications as the basis for each apfi reconfiguration[4]. Then, the requirement class for the mapp reconfiguration is presented.

B. Requirement Class for the Fusion of Particular Indications

The requirement class related to the fusion of particular indications describes transformations of the type

$$f_{fusion} : \{z_{I,i}\} \rightarrow \{z_{CI,x}\} \tag{1}$$

within a single module. Using an if-then-else construct, the requirement class can be expressed as follows[5]. Values that need to be specialised during instantiation are coloured and marked with #.

REQ - fusion of particular indications
Let $\mathbf{Z}_{I,\#nameCI\#,(\#M(x)\#)} := \{z_{I,\#nameI\#}\}_{(\#M(x)\#)}$.
If there is a $z_{I,i,(\#M(x)\#)} \in \mathbf{Z}_{I,\#nameCI\#,(\#M(x)\#)}$ with $z_{I,i,(\#M(x)\#)} = true$ for #confirmTime# ms,
then $z_{CI,\#nameCI\#,(\#M(x)\#)} = true$.
Else, $z_{CI,\#nameCI\#,(\#M(x)\#)} = false$.
EndOfREQ

Here, $\mathbf{Z}_{I,\#nameCI\#,(\#M(x)\#)}$ is the set of all particular indications $z_{I,\#nameI\#,(\#M(x)\#)}$ that are mapped to the categorical indication $z_{CI,\#nameCI\#,(\#M(x)\#)}$ within a specific module $\#M_{(x)}\#$.

The basic pattern for the instantiation condition related to the requirement class is illustrated in Fig. 6(a) using an UML syntax[6]. The pattern corresponds to the generic structure of f_{fusion} extended by the confirm time.

Specific instances of this pattern are searched for in the generated apfi design models, which result from the Axx subprocess. These models comprise the information about every existing instance of f_{fusion} for a specific apfi. Figure 6(b) exemplifies the relevant part of the design model for $cpm_{(R,1)}$. The matches of the instantiation pattern are outlined with a dashed line.

All matches with the same categorical indication are grouped at which each group triggers the instantiation of a corresponding requirement. With regard to the example in Fig. 6(b), two requirement instances are generated. The requirement instance or its textual representation[7] for the categorical indication $z_{CI,BIT}$[8] is as follows:

REQ - fusion of particular indications
Let $\mathbf{Z}_{I,BIT,(cpm(R,1))} := \{z_{I,Partition(1)Fail}, z_{I,CrossCompareFail}\}_{(cpm(R,1))}$.
If there is a $z_{I,i,(cpm(R,1))} \in \mathbf{Z}_{I,BIT,(cpm(R,1))}$ with $z_{I,i,(cpm(R,1))} = true$ for 20 ms,
then $z_{CI,BIT,(cpm(R,1))} = true$.
Else, $z_{CI,BIT,(cpm(R,1))} = false$.
EndOfREQ

[3]The apfi specific subset is defined in the high-level system design model for the Axx subprocess.

[4]Although, strictly speaking, this requirement class does not describe a reconfiguration, it serves to clarify basic mechanisms.

[5]In addition to the actual requirement text, a requirement class comprises further attributes, such as a rationale, assumptions or the allocation to apfi functions assigned to a specific Design Assurance Level (DAL).

[6]For the implementation into the xAx tool suite, the patterns are modelled using a dedicated domain-specific language. Details can be found in [11].

[7]The formal representation of the requirement classes is not considered in this paper but will be part of future publications.

[8]$z_{CI,BIT}$ leads to the execution of a module built-in test (BIT). If a fatal failure is found during this BIT, the module is passivated for the rest of the mission.

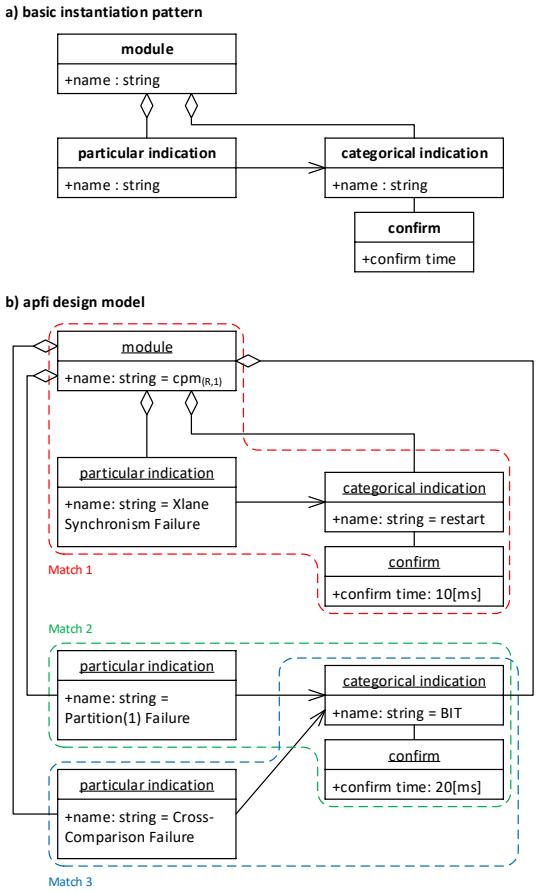

Fig. 6. (a) Basic instantiation pattern for the requirement class of the fusion of particular indications as well as (b) the relevant part of an exemplary apfi design model.

C. Basics of the Reconfiguration Requirement Classes

Compared to the requirement class of the fusion of particular indications, the following has to be considered for the reconfiguration requirement classes. While all reconfigurations are based exclusively on categorical indications, these categorical indications are only software-based quantities. However, states such as a power-off of all lanes of a module cannot be detected by the affected module and thus cannot be expressed by its categorical indications. For this reason, the physical health state z_s of each apfi hardware resource is introduced. Within the scope of the specification, the categorical indications are extended with these physical health states to so-called extended categorical indications. The vector of the extended categorical indications is defined as $\underline{z}_{ECI} := (\underline{z}_s, \underline{z}_{CI})^T_{ECI}$.

Based on the presented reconfiguration principle and the extended categorical indications, all possible system-level reconfigurations of each apfi can be generically described by the transformation:

$$f_{reconfig} : \left\{ \begin{matrix} \underline{s}_{apfi} \\ \begin{pmatrix} \underline{z}_s \\ \underline{z}_{CI} \end{pmatrix}_{ECI} \end{matrix} \right\} \rightarrow \{\underline{s}_{apfi}\}. \qquad (2)$$

Consequently, $f_{reconfig}$ is the basis for all reconfiguration requirement classes at the system level. As an example, the requirement class related to the mapp reconfigurations of apfis with two mapps and up to four cpms is presented in the following section.

D. Requirement Class for the Mapp Reconfiguration

The requirement class in Fig. 7 describes mapp reconfigurations due to the passivation or activation of cpms[9]. Therefore, it enables the specification of each transition from a mapp allocation for q correct cpms to the mapp allocation for r correct cpms. A correct cpm implies that the membership is given for this cpm. For a passivation transition, there is $q \in \{2,3,4\}$ and $r = q-1$. In contrast to this, for an activation transition $q \in \{1,2,3\}$ and $r = q+1$.

To express the requirement class based on $f_{reconfig}$, the vectors $\underline{s}_{xcpm,mapp,q}$ and $\underline{s}_{xcpm,mapp,r}$ as well as the vector $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$ are introduced.

- $\underline{s}_{xcpm,mapp,q}$ describes the initial steady-state apfi configuration, i.e. the apfi-wide mapp allocation for q correct cpms.
- Starting from q correct cpm, $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$ describes the apfi wide view of the passivation or activation of an arbitrary cpm to r correct cpms.
- $\underline{s}_{xcpm,mapp,r}$ describes the resulting steady-state mapp allocation for r correct cpms.

Using these vectors, the mapp reconfiguration requirement class is illustrated in Fig. 7. The used variables can be described as follows:

- $s_{mapp,(cpm(i))}$ describes the mapp currently allocated to an arbitrary $cpm_{(i)}$, with $s_{mapp,(cpm(i))} \in \{mapp_{(1)}, mapp_{(2)}, nil\}$. Nil is a state in which $cpm_{(i)}$ failed passive and thus no mapp is allocated.
- $z_{s,(cpm(j))}$ is the physical health state of an arbitrary $cpm_{(j)}$. It applies that $z_{s,(cpm(j))} \in \{c, f_p\}$, where c is a correct state of $cpm_{(j)}$ and f_p is a state in which $cpm_{(j)}$ failed-passive. Accordingly, $z_{s,(cpm(j))}$ is used to describe the passivation or activation of a $cpm_{(j)}$.
- Such a passivation or activation can be monitored by other, correct $cpm_{(i)}$ using the categorical indication $z_{CI,comm,(cpm(i),cpm(j))}$. It describes the opinion of $cpm_{(i)}$ about $cpm_{(j)}$ regarding their overall communication, with $z_{CI,comm,(cpm(i),cpm(j))} \in \{ok, \neg ok\}$. For example, if a $cpm_{(j)}$ failed-passive no messages are transmitted via any communication network to other $cpm_{(i)}$ which thus set $z_{CI,comm,(cpm(i),cpm(j))} = \neg ok$.

For the indices that define the assignment of these variables to the cpms the following applies.

- Considering $\underline{s}_{xcpm,mapp,q}$, the network side indices $x, y \in \{R, B\}$, with $x \neq y$. For the corresponding module ID indices, there is $i, j, k, l \in \{1, 2\}$, with $i \neq j$ and $k \neq l$.
- Analogously, considering $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$ and $\underline{s}_{xcpm,mapp,r}$, the network side indices $a, b \in \{R, B\}$,

[9]Note that the temporal aspects of this requirement class are neglected in the context of the paper.

REQ - Mapp reconfiguration due to a change from $\#q\#$ to $\#r\#$ correct cpms

$$\begin{pmatrix} \#\underline{s}_{xcpm,mapp,q}\# \\ \#\underline{z}_{ECI,xcpm,mapp,q\mapsto r}\# \end{pmatrix} \mapsto (\#\underline{s}_{xcpm,mapp,r}\#)$$

$$= \left(\begin{array}{c} \#\begin{pmatrix} s_{mapp,(cpm(S(x),i))} & s_{mapp,(cpm(S(x),j))} \\ s_{mapp,(cpm(S(y),k))} & s_{mapp,(cpm(S(y),l))} \end{pmatrix}_q \# \\ \# \left(\begin{array}{cc} \begin{pmatrix} z_{s,(cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(a),m),cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(a),m),cpm(S(b),o))} \\ z_{CI,comm,(cpm(S(a),m),cpm(S(b),p))} \end{pmatrix} & \begin{pmatrix} z_{s,(cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(a),n),cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(a),n),cpm(S(b),o))} \\ z_{CI,comm,(cpm(S(a),n),cpm(S(b),p))} \end{pmatrix} \\ \begin{pmatrix} z_{s,(cpm(S(b),o))} \\ z_{CI,comm,(cpm(S(b),o),cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(b),o),cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(b),o),cpm(S(b),p))} \end{pmatrix} & \begin{pmatrix} z_{s,(cpm(S(b),p))} \\ z_{CI,comm,(cpm(S(b),p),cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(b),p),cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(b),p),cpm(S(b),o))} \end{pmatrix} \end{array} \right)_{q\mapsto r} \# \end{array} \right) \mapsto \left(\# \begin{pmatrix} s_{mapp,(cpm(S(a),m))} & s_{mapp,(cpm(S(a),n))} \\ s_{mapp,(cpm(S(b),o))} & s_{mapp,(cpm(S(b),p))} \end{pmatrix}_r \# \right)$$

EndOfREQ

Fig. 7. Requirement class for the mapp reconfiguration due to the passivation or activation of cpms.

with $a \neq b$. Furthermore, the related module ID indices $m, n, o, p \in \{1, 2\}$, with $m \neq n$ and $o \neq p$.

There is a major difference between the instantiation of the requirement class for the fusion of particular indications and the requirement class for the mapp reconfiguration. Here, the aforementioned mapp allocation rules for apfis with two mapps and up to four cpms are a part of the requirement class. All correct cpms have to meet these rules, which are as follows:

1) The number of replicas of $mapp_{(i)}$ per network side is at most 1: $i \in \{1, 2\}, x \in \{R, B\} : N_{mapp(i),(S(x))} \leq 1$.
2) The number of replicas of $mapp_{(1)}$ is equal to or at most greater than 1 compared to the number of replicas of $mapp_{(2)}$: $N_{mapp(1)} - 1 \leq N_{mapp(2)} \leq N_{mapp(1)}$.

The first rule is based on a design decision related to the communication architecture of the apf. The second rule ensures that each mapp is executed for as long as possible while taking into account the mapp prioritisation.

Using these rules, the requirement class itself generically describes each mapp reconfiguration due to a passivation or activation. Consequently, the number of mapps and cpms is the only characteristic required for the generation of the requirement instances. The associated basic instantiation pattern is shown in Fig. 8(a). If an apfi does show this characteristic, all requirement instances of this class are generated, i.e. one requirement instance for each possible specific activation and passivation transition for the apfi specific number of cpms and mapps. The instantiation is illustrated in the following.

The apfi design model contains the information about the number of mapps and cpms of an apfi. An exemplary model for an apfi with two mapps and four cpms is illustrated in Fig. 8(b). The depicted mapp to cpm relation results in a single match for the class's instantiation pattern. Accordingly, requirement instances for the activation transitions $q \mapsto r \in \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 4\}$ have to be generated. Furthermore, requirement instances for the passivation transitions $q \mapsto r \in \{4 \mapsto 3, 3 \mapsto 2, 2 \mapsto 1\}$ have to be generated. In the following, the instantiation steps carried out for each requirement instance are described using the example of a passivation transition from $q = 4$ to $r = 3$ correct cpms.

First, the value of $\underline{s}_{xcpm,mapp,q}$ is determined based on $q = 4$ and the apf generic mapp allocation rules. The resulting

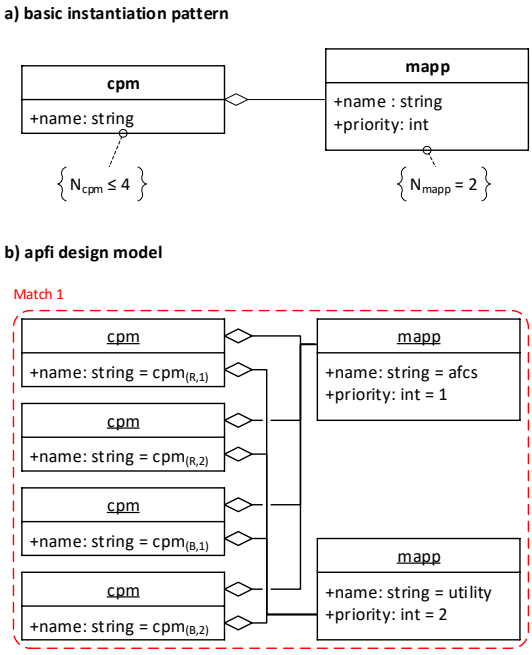

Fig. 8. (a) Basic instantiation pattern for the requirement class of the mapp reconfiguration as well as (b) the relevant part of an exemplary apfi design model.

correct mapp allocation for four correct cpms is

$$\underline{s}_{xcpm,mapp,4} = \begin{pmatrix} mapp_{(1)} & mapp_{(2)} \\ mapp_{(1)} & mapp_{(2)} \end{pmatrix}_4 . \tag{3}$$

Second, the apfi-wide view on the passivation of a cpm is expressed using $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$. In our example, the passivation of an arbitrary $cpm_{(S(b),p)}$ is represented by

$$\underline{z}_{ECI,xcpm,mapp,4\mapsto 3} = \left(\begin{array}{cc} \begin{pmatrix} c \\ ok \\ ok \\ \neg ok \end{pmatrix} & \begin{pmatrix} c \\ ok \\ ok \\ \neg ok \end{pmatrix} \\ \begin{pmatrix} c \\ ok \\ ok \\ \neg ok \end{pmatrix} & \begin{pmatrix} f_p \\ - \\ - \\ - \end{pmatrix} \end{array} \right)_{4\mapsto 3} . \tag{4}$$

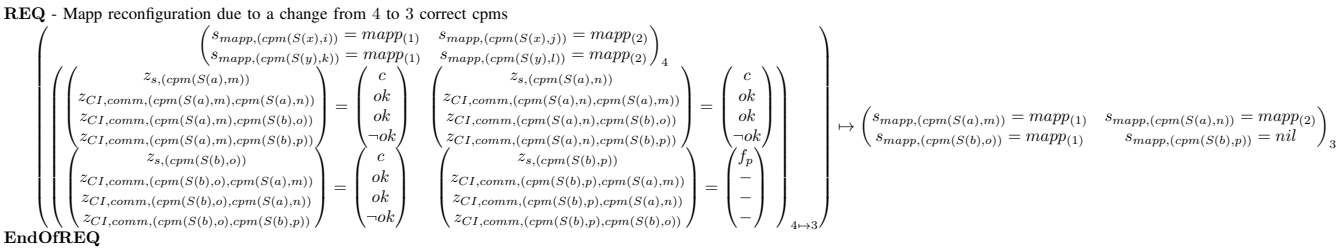

Fig. 9. Requirement instance for the mapp reconfiguration due to the passivation of an arbitrary cpm, from $q = 4$ to $r = 3$ correct cpms.

Third, with $r = q - 1$ the value of $\underline{s}_{xcpm,mapp,r}$ is determined based on the aforementioned mapp allocation rules, i.e.

$$\underline{s}_{xcpm,mapp,3} = \begin{pmatrix} mapp_{(1)} & mapp_{(2)} \\ mapp_{(1)} & nil \end{pmatrix}_3 . \tag{5}$$

The resulting requirement instance for the passivation of an arbitrary cpm from one of $q = 4$ correct cpms is shown in Fig. 9.

Note that a single requirement instance specifies the passivation transition from $q = 4$ to $r = 3$ correct cpms for all possible permutations. Here, a permutation is a specific configuration of how the mapps are allocated to the cpms of an apfi as well as of the cpm that failed-passive. Thus, a permutation is a specific selection of the indices that define the assignment to the cpms[10].

Regarding the verification of such a requirement instance, the following has to be considered. First, the transition must be verified for all specific permutations for a complete verification of the required behaviour. Second, to our understanding, performing the related tests based on the stimulation of the categorical indications is not sufficient for a verification at the system level. In order to verify the entire end-to-end system behaviour, the composed transformation

$$f_{moni} \circ f_{fusion} \circ f_{reconfig} \tag{6}$$

is considered, with $f_{moni} : \{ev_i\} \rightarrow \{z_{I,i}\}$. This means that to verify a reconfiguration, not the categorical indication is stimulated but an actual exemplary failure event associated with the categorical indication. In the given example, such a failure event could be the power interruption of a cpm.

V. Conclusion

For a significant reduction of the costs related to the development of fault-tolerant real-time avionics systems, the Flexible Avionics Platform (apf) and an associated comprehensive automation process, the AAA process, were developed. The apf is characterised by a high-level abstraction layer, the Platform Management, providing a full abstraction of distribution, fault-tolerance, and redundancy towards integrated system functions. The associated AAA process combines the high flexibility of this platform-based approach with a high degree of automation. Therefore, the generation of the development and qualification artefacts of each Flexible Avionics Platform Instance (apfi), such as the system-level specification and the verification artefacts, is largely automated.

This paper focused on the automatic generation of the system-level specification for the high-level Platform Management. The presented approach with its specification subdomains is made possible by the systematic and clear structuring of the apf [15]. Each of the named subdomains is dedicated to specific apfi behaviour. These are the intra-module reconfigurations, inter-module reconfigurations, peripheral reconfigurations, and an apfi's entire operation moding. Our article describes the definition of the requirement classes for the apfi reconfigurations which can be expressed based on an apf generic set of categorical indications that classify all possible failure events of an apfi. On this basis, a limited number of requirement classes were defined. They are sufficiently generic to cover the apf's entire usage domain while still being expressive and comprehensible. This facilitates a manual validation as part of a certification process. In addition to the presented actual requirement text of a requirement class, other attributes can of course be added according to the applied requirements standard. The developed requirement classes for the high-level Platform Management were integrated into the AAA tool suite and thus enable the automatic instantiation of reconfiguration and operation moding requirements. This further completes our overall approach for a cost-efficient development and qualification of fault-tolerant, distributed and integrated avionics systems such as fly-by-wire systems.

The apf approach was successfully demonstrated within multiple projects up to an in-flight demonstrator featuring automatic take-off and landing [17], [18], [19]. Our ongoing research in the LuFo V-3 research project Secured System for Manned Multicopter (SESYMM) focuses on optimising the generated artefacts with regard to DAL A conformity. Moreover, the AAA tool suite is used for the development of a fly-by-wire platform for a remotely piloted aircraft system based on a CS-23 aircraft for operation in the non-segregated airspace without an onboard safety pilot.

Glossary

apf The Flexible Avionics Platform is a platform-based development approach featuring an integrated, distributed, and highly redundant architecture. It is characterised by a high-level abstraction layer, the

[10]The passivation transition illustrated in Fig. 4 is a specific permutation of this requirement instance, with $x = R$, $y = B$, $i = k = 1$, $j = l = 2$, $a = B$, $b = R$, $m = p = 1$, and $n = o = 2$.

Platform Management, enabling integrated applications to be executed in a failure-free virtual simplex environment. Thus, applications can be reduced to their cybernetic control law.

apfi An instance of the Flexible Avionics Platform.

cpm Core processing modules plan and execute the central management decision within an apfi. In addition, they execute the applications for the integrated system function laws.

mapp A multi-application comprises several integrated system function laws of the same criticality level. A dynamic in-flight mapp reallocation between cpms enables an efficient use of the cpm hardware.

REFERENCES

[1] J.-B. Itier, "A380 integrated modular avionics—the history, objectives and challenges of the deployment of ima on a380," in *Proceedings of the ARTIST2 Meeting on Integrated Modular Avionics, Roma, Italy*, 2007, pp. 12–13.

[2] B. Annighöfer and E. Kleemann, "Large-scale model-based avionics architecture optimization methods and case study," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 6, pp. 3424–3441, 2019.

[3] T. Gaska, C. Watkin, and Y. Chen, "Integrated modular avionics-past, present, and future," *IEEE Aerospace and Electronic Systems Magazine*, vol. 30, no. 9, pp. 12–23, 2015.

[4] B. Kornek-Percin, B. Petersen, M. Reichle, and J. Bader, "New ima architecture approach based on ima resources," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 2015, pp. 6A2–1.

[5] M. Halle and F. Thielecke, "Evaluation of the ashley seamless tool-chain on a real-world avionics demonstrator," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 2017, pp. 1–9.

[6] B. Annighoefer, M. Brunner, J. Schoepf, B. Luettig, M. Merckling, and P. Mueller, "Holistic ima platform configuration using web-technologies and a domain-specific model query language," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–10.

[7] J. Yin, B. Lawler, and H. Jin, "Application of model based system engineering to ima development activities," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 2017, pp. 1–7.

[8] S. Görke, *Eine flexible Plattform für Fly-by-Wire-Systeme-Spezialisierbarkeit als Schlüssel zur effizienten Entwicklung sicherheitskritischer Avionik*. Verlag Dr. Hut, 2013.

[9] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[10] F. Kraus, *Verfahren zur weitgehend automatisierten Erzeugung der Middleware für hoch ausfallsichere, integrierte Avioniksysteme mittels Model-Integrated Computing*. Verlag Dr. Hut, 2018.

[11] T. Belschner, *A Method for the Automated Generation of Requirements and Traceability for a Distributed Avionics Platform*. Verlag Dr. Hut, 2020.

[12] P. Müller, *Automated Test Artifact Generation for a Safety-Critical Integrated Avionics Platform*. Verlag Dr. Hut, 2021.

[13] C. Block, S. Dikmen, and R. Reichel, "Automated test case generation for the verification of system and high-level software requirements for fly-by-wire platforms," in *AIAA SCITECH 2022 Forum*, 2022, p. 0254.

[14] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Cornell University, Tech. Rep., 1994.

[15] T. Hoffmann, R. Wipperfürth, and R. Reichel, "Enabling the automated generation of the failure and redundancy management for distributed and integrated fly-by-wire avionics," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. IEEE, 2021, pp. 1–10.

[16] F. Cake, *Supervisor für eine komplexe verteilte Avionikplattform*. Verlag Dr. Hut, 2016.

[17] L. Dalldorff, R. Luckner, and R. Reichel, "A full-authority automatic flight control system for the civil airborne utility platform s15-lapaz," *Euro GNC*, 2013.

[18] R. Kueke, P. Mueller, S. Polenz, R. Reichel, F. Pinchetti, J. Stephan, A. Joos, and W. Fichter, "Fly-by-wire for cs23 aircraft-core technology for general aviation and rpas," in *Aviation in Europe Innovation for Growth-7th European Aeronautics Days*, 2015.

[19] S. Görke, R. Riebeling, F. Kraus, and R. Reichel, "Flexible platform approach for fly-by-wire systems," in *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*. IEEE, 2013, pp. 2C5–1.