From Architecture Models to a Target Platform: A Systematic Approach to Model-Driven Development of Dynamically Reconfigurable Avionics Systems

Bojan Lukić
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
Email: bojan.lukic@dlr.de
0009-0002-4286-1901

Phillip Nöldeke
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
Email: phillip.nöldeke@dlr.de
0009-0008-1537-2890

Umut Durak
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
Email: umut.durak@dlr.de
0000-0002-2928-1710

Matthias Klimmek
Software Engineering
SYSGO GmbH
Klein-Winternheim, Germany
Email: matthias.klimmek@sysgo.com

Simon Jakob
Digital Content Management
SYSGO GmbH
Klein-Winternheim, Germany
Email: simon.jakob@sysgo.com
0009-0009-0969-8736

Martin Gläser
Verification Engineering
SYSGO GmbH
Rostock, Germany
Email: martin.glaeser@sysgo.com

Abstract*—Integrated Modular Avionics (IMA) is an advanced avionics architecture with a unified design, standardized components, and a unique computing paradigm. By allocating multiple functions to single processing units, IMA brought breakthroughs regarding efficiency, modularity, and maintainability to the aviation industry. The innovations in avionics, such as IMA, are ever-growing, with one of the more recent ones being dynamically reconfigurable avionics. Dynamically reconfigurable avionics refers to the state, i.e., configuration, change of the avionics system while it is executing. This allows the system to adapt and rearrange its software partitions and resource utilization as needed during runtime. By enabling the system to dynamically allocate and reallocate available resources relating to processing power, memory, and I/O devices, it allows for adapting to changing operational requirements and handling failures. This ensures the functional system availability for a longer time. The study introduced in this paper presents a model-driven methodology that addresses the complexity of developing such advanced avionics systems. The paper consists of three major aspects with regards to developing dynamically reconfigurable avionics: the architecture specification, model-to-text transformation for code and configuration generation, and embedding on a target platform. The paper demonstrates a systematic approach for capturing all essential architecture parameters of an avionics configuration followed in a model-driven approach. This approach can significantly enhance the development of dynamically reconfigurable avionics systems.

***Index Terms*—Avionics, Dynamic Reconfiguration, ARINC 653, Model-Driven Development, RTOS, Hypervisor**

I. Introduction

Integrated Modular Avionics (IMA) systems were initially designed to consolidate hardware resources and improve avionics architectures as opposed to former federated systems. IMA systems have undergone significant evolution to accommodate more advanced technologies and applications. The ARINC 653 standard, for instance, introduced a robust application executive for advanced and complex software systems. This established the space, time, and interface partition for a more efficient resource usage in avionics systems with real time and safety-critical needs. With the advent of multi-core processors, larger field programmable gate arrays, and increasingly complex network protocols, the capabilities and functionalities embedded within IMA systems are ever-growing. Applications become more sophisticated and demanding, requiring not only larger computational resources but also robustness, often adding conflicting demands.

Despite these requirements, the configuration of applications within IMA systems remains static which limits the ability to adapt to fluctuating demands, potential system failures, and increases system availability. To address these constraints, dynamic reconfiguration within IMA systems presents a promising solution. This approach, however, introduces new challenges. These are, for instance, the need to verify all reachable states of a reconfigured system and manage the transition mechanisms between configurations effectively.

Model-driven development emerges as a strategy for addressing these challenges. It supports the planning phase, aids in optimizing partitioning policies and failure modes, and facilitates the verification of configuration variants. This methodology is particularly relevant given the complexity and safety-critical nature of aviation systems [1, 2, 3].

The objective of this paper on the one hand is to provide a clear understanding of dynamic reconfiguration in the context of ARINC 653. On the other hand, the paper illustrates a model-driven methodology for implementing avionics configurations for a dynamically reconfigurable end system. Through this exploration, the authors aim to lay the groundwork for more adaptive, efficient, and safe avionics systems.

The remaining work is structured as follows: Section II covers essential background to lay the foundation for the main part of the paper. Section III presents related work in the field of reconfigurable avionics and its development practices. The development of an avionics configuration is demonstrated in section IV with a practical use case. Sections V and VI discuss and conclude the work, respectively.

II. Background

Significant for this work are the guidances, tools, and definitions for the development of dynamically reconfigurable avionics. The following subsections cover following topics: First, the fundamental definitions of elements found in avionics and the context of reconfiguration. Second, the essential guidances discussing configurations and general avionics development considerations. Third, the tools used for the use case described in the main part. And finally, the theory and considerations for dynamically reconfigurable avionics.

A. Definitions

The following paragraphs specify essential definitions which are frequently used in later parts of this paper. The definitions are direct quotes or paraphrased definitions taken from [4], [5], and [6].

Integrated Modular Avionics
IMA is a shared set of flexible, reusable, and interoperable hardware and software resources that, when integrated, form a platform. This platform provides services, designed and verified to a defined set of safety and performance requirements, to host applications performing aircraft functions.

Module
A component or collection of components that may be accepted by themselves or in the context of IMA. A module may also comprise other modules. A module may be software, hardware, or a combination of hardware and software, which provides resources to the IMA-hosted applications. Modules may be distributed across the aircraft or may be co-located.

Function
The capability of the aircraft that may be provided by the hardware and software of the systems on the aircraft. Functions include flight control, autopilot, braking, fuel management, flight instruments, etc. IMA has the potential to broaden the definition of avionics to include any aircraft function.

Application
Software and/or application-specific hardware with a defined set of interfaces that, when integrated with a platform, performs a function.

Configuration
The composition of module parameters in form of a specification. In this work, the parameters are narrowed down to memory, period, duration, port attributes, and processor core requirements for partitions mapped onto modules. A configuration can be used to initialize an avionics system.

Reconfiguration
A change and/or an update to a system configured to a particular system state. The purpose of a reconfiguration is to add, remove, or replace functionality on the system. The sum of all the changes is the reconfiguration from one system state to another.

Dynamic/Runtime Reconfiguration
Similarly to the previous definition, dynamic or runtime reconfiguration is a change and/or an update to a system configured to a particular system state. Additionally, when performing dynamic reconfiguration, the system is not stalled when changing its configuration. That means that the configuration of the system is changed during *runtime*, that is, while the system is executing. Refer to II-D for a more detailed explanation on dynamic reconfiguration. Note that for this work the terms *dynamic reconfiguration* and *runtime reconfiguration* bear the same meaning. They are therefore used interchangeably.

B. Standards, Guidelines, and Guidances

The documents described in this subsection revolve around standards and guidelines for the general development of avionics systems. The selection depicts the most relevant references for integrated avionics systems. If applicable, the paragraphs go into specifics about avionics configuration and dynamic reconfiguration. The use case described in this paper is developed under consideration of these references.

ARINC 653
ARINC 653 defines a specification for the development of an application executive with real-time and fault recovery properties for software execution. The standard defines a partitioned operating system environment for running multiple software applications, referred to as partitions, on a single hardware platform. Each partition is isolated, ensuring that any failures or issues it experiences do not affect the operation of other partitions.

While not specifically addressing the topic, ARINC 653 does provide some mechanisms that can be used to achieve dynamic reconfiguration: The standard defines partition management services, such as create, start, stop, and delete, that can be used to dynamically reconfigure the system. Moreover, it specifies a resource allocation mechanism that allows partitions to request and release resources dynamically. This enables the system to adapt to changing requirements and allocate resources more efficiently. ARINC 653 defines mode management services that enable the system to switch between different operational modes. Although not directly related to dynamic reconfiguration, mode management could be used to change the system's behavior in response to changing conditions.

As for avionics configurations, ARINC 653 recommends using a standardized format for representing configuration data, which can be serialized and exchanged between systems. Such a standardized format is demonstrated with a configuration in XML format, covering partitioning, schedules, and health monitoring of a processing module [5]. The end system PikeOS, used for the demonstrator in this work, is fully compliant with ARINC 653.

DO-297/ED-124

The DO-297/ED-124 standard outlines the processes, activities, and artifacts that should be followed during the development lifecycle of IMA systems. It covers various aspects, including requirements engineering, software design, verification and validation, configuration management, and system integration. The standard also provides guidelines for the certification of IMA systems, which involves demonstrating compliance with regulatory requirements and safety objectives.

As for avionics systems configuration, DO-297/ED-124 expects there to be different configurations for IMA systems that include "configuration data for scheduling applications and allocating resources. This data could be implemented as a set of configuration tables or files established at build time or as separately loadable data" [4].

DO-297/ED-124 explicitly considers reconfiguration capabilities of IMA systems. Section 3.7.1.1 about IMA system configuration data states that "[c]onfiguration data [...] is used by the IMA system to: [...] **activate** or **deactivate** modules, resources, or functions, e.g., adaptation of the software to several aircraft configurations using option-selectable software or data." Further, the guidance states that "[o]ther types of configuration data may be used by the hosted applications on the IMA system to: [..] **activate** or **deactivate** functions or application-specific resources (e.g., adaptation of the software to several aircraft configurations using option-selectable software, of the hosted applications) [and] adapt the application to the aircraft configuration" [4].

Therefore, both module as well as application reconfiguration is supported by DO-297/ED-124. That is, the activation or deactivation of modules or applications specified in a configuration. Moreover, even though it does not explicitly write about it, the guidance does not rule out reconfiguration during runtime. PikeOS is compliant with the DO-297/ED-124 standard.

AC 20-170 / AMC 20-170

The AC 20-170 / AMC 20-170 advisory refers to the Integrated Modular Avionics (IMA) guidance provided by the European Union Aviation Safety Agency and Federal Aviation Administration.

For the work at hand, AC 20-170 / AMC 20-170 plays a significant role when generating tool-driven configurations for some IMA system. The tool quality needs to be ensured. These tools can potentially produce an erroneous configuration which might not be easily detectable during subsequent integration and verification steps in the engineering process.

The guidance states that any parameter data items used for configuration "need to be defined, managed and documented at the appropriate level (platform, module, application) and to comply with the AMC 20-115() guidance" [7].

STANAG 4626 / DIN EN 4660

The STANAG 4626 / DIN EN 4660 standard is part of a series of standards called *Aerospace series – Modular and Open Avionics Architectures*. These standards describe a set of standards, terms and guidelines for open A3 (Advanced Avionic Architectures) architectures in the aerospace industry. The three main goals for the MOAA Standards are reduced life cycle costs, improved mission performance, and improved operational performance.

Part 5 of STANAG 4626 / DIN EN 4660 refers to ASAAC2-GUI-32450-001-CPG Issue 01 for specifics about system configuration and dynamic reconfiguration [8]. The ASAAC guideline states that "the process of reconfiguration is predictable, going from one analysable sub-state to the next by the application of actions." Further, "[reconfigurations] are held in the blueprints, and are designed to ensure that the reconfiguration changes are performed in a predictable fashion" [6]. The phrasing gives reasons to presume that STANAG 4626 / DIN EN 4660 proposes fixed schedules for reconfiguration which are defined at design time. Similar to DO-297/ED-124, STANAG 4626 / DIN EN 4660 does not rule out dynamic reconfiguration.

C. Tools

The software used for the developed setup are MathWorks' MATLAB, System Composer, and Simulink, as well as SYSGO's CODEO and PikeOS. Additionally, the ARINC 653 blockset add-on for Simulink is used for custom software integration and system image generation.

ARINC 653 Blockset

The ARINC 653 blockset is a custom set of Simulink blocks developed by The MathWorks. The blockset uses Simulink blocks to represent a subset of the ARINC 653 services. They allow a user to model, simulate and generate code for an ARINC 653 partition in Simulink. Each block contains a user interface dialog box for defining parameters for the ARINC 653 service. For example, the `READ_SAMPLING_PORT` dialog enables specification of port name, maximum size and refresh period. The generated code is compliant with the ARINC 653 Application Program Interfaces and can be compiled and linked with an ARINC 653-compliant operating system.

In addition to the custom ARINC 653 blocks, the blockset includes a utility for creating skeleton Simulink models based on an ARINC 653 XML file. A skeleton model is created for each partition. Each model contains Simulink blocks to create the processes for the partition. Each process contains the blocks for accessing data via sampling ports and/or queueing ports. The blocks contain the configuration information defined in the XML file.

PikeOS

PikeOS is both a real-time operating system and a bare-metal type 1 hypervisor[1], developed by the embedded software company SYSGO. It is designed to fulfill hard real-time requirements in safety-critical environments such as those in aviation. The hypervisor functionality allows partitions, also called *personalities* in the PikeOS context, to run concurrently and safely on the same hardware within a PikeOS execution environment. These partitions can be simple applications but also guest operating systems.

Because of the safety approach of PikeOS, the configuration of partitions must be planned ahead and cannot be changed during runtime in terms of creating or deleting new partitions. PikeOS requires a master partition to be set up with the information which partitions run concurrently. This ensures determinism which is pivotal to any safety-critical application. However, it is possible to start or stop a partition during runtime to restart certain applications or reduce CPU load from unrequired tasks. Partitions can also be configured in idle mode so they do not automatically start during system boot but only on request.

CODEO

CODEO is an Eclipse-based Integrated Development Environment (IDE) supporting the programming languages C and C++, developed by SYSGO. The CODEO IDE includes project management, code browser, configuration management, and interface components that can be further extended by other Eclipse plug-ins.

Via the project manager feature of the IDE, instances of PikeOS can be set up. The configuration manager contains a graphical configuration editor and an integrity checker which prevents invalid configurations. Partitions can then be added or removed and applications as well as services like drivers, stacks, and I/O servers are set up. It also provides graphical tools to configure multiple partition time schedule schemes and connect sampling or queueing ports between partitions.

Generated configurations are stored in XML format which provides both a user readable file and the possibility for automated changes and additions. Any application running on PikeOS can be debugged independently of all other concurrent applications. Several applications can be debugged at the same time. Applications developed with CODEO are deployed directly on the target hardware or the QEMU emulation environment.

[1]As explained in: Robert P. Goldberg. "Architectural Principles for Virtual Computer Systems."[9].

D. Dynamic Reconfiguration

This subsection aims at laying the theoretical foundation of dynamic reconfiguration for avionics. The scheduling of partitions is a fundamental functionality that is necessary to understand the principle of dynamic reconfiguration. Fig. 1 shows an exemplary schedule for three partitions managed by a hypervisor inside a General Processing Module (GPM). The duration a single partition is running within one periodic cycle is called partition (time) window. All partition windows combined make up the major frame. Suppose *Partition 1* has a duration of $100\,\mathrm{ms}$ and *Partition 2* and *Partition 3* have a duration of $60\,\mathrm{ms}$ each. To keep the example simple, one can assume that there is no slack between the partitions and each partition runs once per major frame. Therefore, the major frame has a duration of $220\,\mathrm{ms}$.

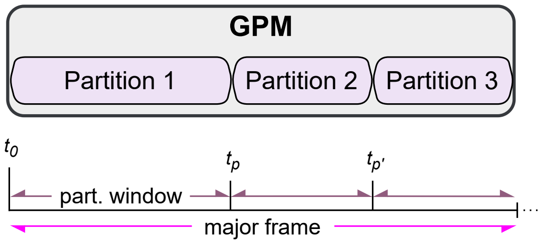

Fig. 1: Exemplary schedule for three partitions running inside a GPM [10]

The hypervisor and Real-Time Operating System (RTOS) PikeOS receives fixed schedules exported from CODEO. This means that spare partitions are not included in the major frame running on PikeOS. A spare partition takes over the partition window from the partition it replaces after reconfiguration.

Fig. 2 shows a minimal setup for dynamic reconfiguration. The GPM contains three partitions, *Partition 4* is a spare partition. After *Partition 3* becomes erroneous, *Partition 4* is activated and takes over the functionality of the erroneous partition. The use case demonstrated in the main part of this paper follows the reconfiguration principle shown in Fig. 2. It is important to note that there are more approaches to reconfiguration. Also, the depiction in Fig. 2 does not consider multi-core systems.

Dynamic reconfiguration can significantly enhance system safety and resource utilization. In terms of safety, by reallocating functionality in the event of faults, the system can maintain operational integrity. In terms of resource utilization, multiplexing applications based on the flight phase allows for more efficient use of hardware as the same resources can be repurposed for different tasks, as needed. Consider a theoretical example with the following assumptions:

- One partition resides on a single core
- A core can host multiple partitions
- Each partition can have multiple tasks
- Partitions run native PikeOS applications
- A software application in each partition contains a rerouting matrix to handle port reconfiguration

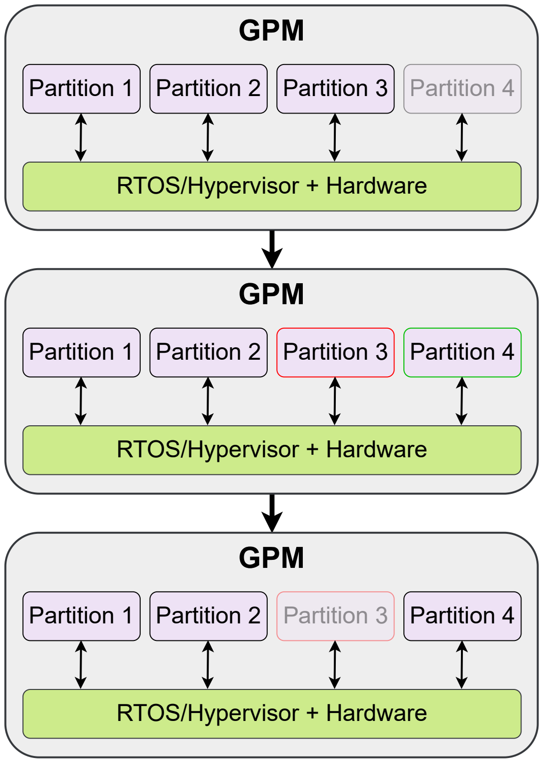

Fig. 2: Exemplary reconfiguration procedure with three common and one spare partition [10]

If a partition is detected as faulty by some health management monitor, a copy of the partition, using different memory space, can be enabled. The rerouting matrix application then reconfigures the connections to the new partition. Reconfigurations are triggered by a health management monitoring system that updates the rerouting matrix and adjusts the connections between partitions. This system continuously monitors the health and status of partitions and initiates reconfiguration as necessary to maintain system integrity and performance.

III. Related Work

Various research in the field of reconfigurable avionics platforms has been conducted in the last decades. The following references were used as a foundation for the use case presented in the paper at hand.

Pagetti et al. [11] provide insights into the safety assessment process of a reconfigurable IMA architecture, emphasizing the importance of formal models in supporting design activities and safety assessments. The authors highlight the use of safety models to guide architecture design, drive safety requirements, and allocate Design Assurance Levels early in the system design phase. Furthermore, the work delves into the development of a safety model to describe both nominal and faulty behavior of reconfiguration mechanisms. It aids in understanding fault propagation among key components of the architecture. The study emphasizes the significance of generating test scenarios from the safety model to guide the testing of reconfiguration mechanism implementations and select relevant test cases for safety assessment. Additionally, the study explores the demonstration of reconfiguration scenarios on the SCARLETT platform, covering various use cases such as reconfiguration on ground and in-flight, as well as reconfiguration at different levels of the system. The authors detail the testing of failure scenarios, including fault simulation methods and observation techniques to assess test outcomes. The study is supported by previous research, such as the one presented in [12]. In summary, the research by Pagetti et al. provides valuable insights into the safety assessment, testing, and demonstration of reconfigurable IMA platforms, highlighting the importance of formal models, safety requirements, and failure scenario testing in ensuring the reliability and effectiveness of reconfiguration mechanisms in avionics systems.

In subsequent related research [13], Durrieu et al. discuss the DREAMS project, supplementary focusing on avionics' middleware role in reconfiguration and adaptation strategies. It presents strong temporal and spatial partitioning for supporting mixed-criticality systems and providing virtualization technologies. Key aspects include time and space partitioning principles, configuration definitions, and memory access regulation for critical tasks. Certifiability for industrial applications is highlighted as a future research direction.

In [14], Zaeske et al. explore dynamic reconfiguration in an ARINC 653 compliant environment for avionics platforms using Artificial Intelligence (AI) applications. The paper highlights the importance of self-supervision in AI systems and the role of ARINC 653 standards in enabling dynamic reconfiguration. Key aspects covered include reconfiguration of partitions, communication channels adaptation, and timing of reconfiguration. The paper emphasizes the need for anomaly detection, data plausibility analysis, and operational constraints to ensure safe and reliable AI-based avionics systems. The authors conclude that dynamic reconfiguration will be a key component for dependable avionics systems. They recommend reconfiguration strategies with a combination of spare partitions with multiple module schedules. The paper refers to multiple other works, such as from Cook [15], Kenneth [16], and López-Jaquero [17]. In [16], the authors emphasize the importance of fault tolerance and reconfiguration in integrated avionics design. They highlight the importance of robust fault tolerance and reconfiguration strategies for maintaining expected service levels and operational capabilities, contributing to overall system reliability and availability. [17] proposes solutions to address limitations regarding fault tolerance in ARINC 653. The authors' suggested approach is the decoupling of partitions and routing of inter-partition communication through a mediator. A shortcoming of the paper is that it does not consider applications that require state.

The paper at hand considers above references for the model-driven development of dynamically reconfigurable ARINC 653-compliant avionics systems. It does not necessarily enhance existing works from these references. This work rather constitutes a proof of concept for the model-driven development of state-of-the-art dynamically reconfigurable avionics systems that are compliant with ARINC 653.

IV. Systematic Framework for ARINC 653-Compliant Avionics Systems Development

The presented framework is an extension of the framework from [18]. It is framed in three different environments: 1. Systems Engineering, 2. Software Integration, and 3. End System Embedment. The major systems engineering frame contains both the software integration as well as the system deployment on an end system.

The aim of the setup is a proof of concept for model-driven development of ARINC 653-compliant dynamically reconfigurable avionics system. The system parameters are captured in a high-level meta model using the model-driven development software System Composer. After creating model instances of the avionics system, configurations are exported in a serialized format. The generated XML configuration file is imported in the ARINC 653 blockset extension for Simulink and system binaries are generated for CODEO. Finally, system images are created in CODEO which are loaded on the PikeOS end system. The system configuration is not tested on any dedicated hardware but rather in an emulated environment. Fig. 3 illustrates the workflow.

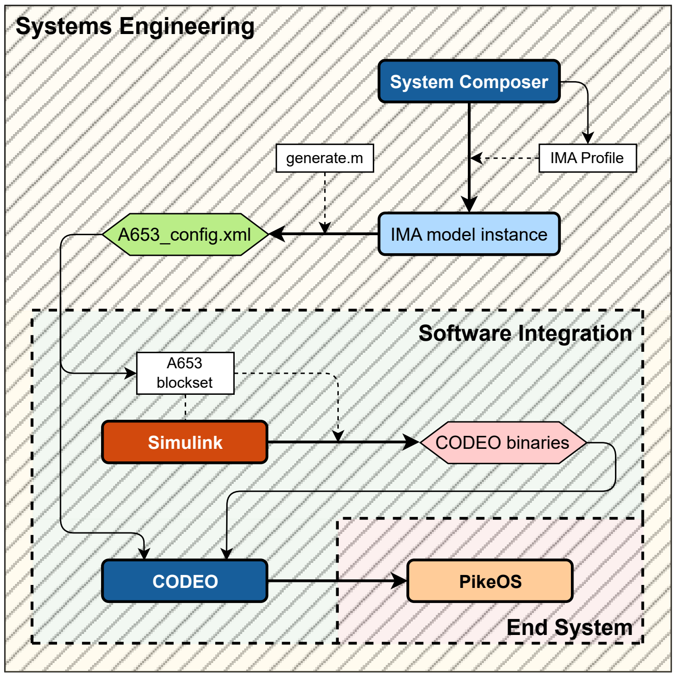

Fig. 3: Workflow followed in this paper portrayed in the systems engineering and software integration landscape

The procedure presented here does not extend existing standards presented in Section II. It is rather demonstrating the state-of-the-art in model-driven development of dynamically reconfigurable avionics systems. That is, the generation of ARINC 653-compliant configurations from high-level models to the point of software integration and the embedding of a configuration on an end system. For this purpose, the use case for an Advanced Flight Assistance System (AFAS) was formulated.

A. *Use Case – Advanced Flight Assistance System*

The proof of concept is realized with AFAS. The system contains different applications which form a safety system to avoid catastrophic collisions with terrain or other aircraft. The system consists of three main applications: 1. Terrain Awareness and Warning System, 2. Collision Avoidance System, and 3. Threat Localization System. The three applications are described in more detail in the following paragraphs.

Terrain Awareness and Warning System (TAWS)

As described in [19]: TAWS is a system with “aural and visual alerts to aid in preventing an inadvertent controlled flight into terrain (CFIT) accident.” TAWS can identify different types of threats. Those include excessive rates of descent, excessive closure rate to terrain, flight into terrain when not in landing configuration, and descent of the airplane to 500 feet above the terrain or the nearest runway elevation with the voice callout “Five Hundred”, to name a few [20].

Collision Avoidance System (CAS)

The Collision Avoidance System (CAS) is “a family of airborne devices that function independently of the ground-based air traffic control (ATC) system, and provide collision avoidance protection for a broad spectrum of aircraft types. All [CAS] systems provide some degree of collision threat alerting, and a traffic display” [21]. “The system will provide appropriate aural and visual advisories to the flight crew to take action to ensure adequate separation when the computer analysis of the intruding aircraft transponder replies predict a penetration of the protected airspace” [22].

Threat Localization System

The Threat Localization System (TLS) deployed in this use case comes in two variations. A TLS using Automatic Dependent Surveillance–Broadcast (ADS-B) and one using Computer Vision (CV), i.e., object detection. ADS-B is “a surveillance system that uses a global navigation satellite system, aircraft avionics, and ground and/or space-based infrastructure to accurately and quickly transmit flight information. This includes aircraft identification, position, altitude, and velocity between aircraft and air traffic control. This signal can be captured on the ground or in space for surveillance purposes (ADS-B-out) or on-board other aircraft for air traffic situational awareness (ADS-B-in) and airborne separation assistance” [23]. The CV use case is described in detail in [24]. In short, two cameras, one on each aircraft wing, are used to perform object detection and calculate the location of intruding aircraft with stereo image depth estimation.

Both the TLS-ADSB and TLS-CV application are used to locate intruding aircraft and send information about those aircraft to the CAS. The CAS then gives advisories to avoid a potential collision. For demonstration, the TLS-ADSB partition is disabled during runtime. The TLS-CV partition is activated at the same time to take over functionality which demonstrates the dynamic reconfiguration capability of the proposed architecture. The details are shown in a later section of the paper.

B. ARINC 653-Compliant System Configuration

Three major requirements can be deduced from the ARINC 653 standard for the development of avionics systems:

1) **Time isolation** – partitions are executed with a temporal isolation
2) **Space isolation** – each partition receives dedicated space in the processing module's memory
3) **Fault coverage** – capturing and recovering faults on the processing module

ARINC 653 specifies three primitive types for a module of a configuration. With their respective parameters they constitute a complete ARINC 653-compliant avionics configuration, covering the three requirements mentioned above:

```
Module
 |__Partition
 |   |__PartitionDefinition – the name and ID
 |   |  of the partition
 |   |__PartitionPeriodicity – the partition
 |   |  periodicity
 |   |__MemoryRegions – the memory region
 |   |  mapped into the partition
 |   |__PartitionPorts – the ports of the
 |      partition
 |
 |__Schedules
 |   |__PartitionTimeWindow
 |
 |__HealthMonitoring
     |__SystemErrors – the list of system
     |  errors
     |__ModuleHM – the module level HM table
     |__MultiPartitionHM – the partition level
     |  HM table for multiple partitions
     |__PartitionHM – the partition level HM
        table
```

In the presented setup, the aforementioned parameters are captured in an avionics profile created with System Composer. The model architecture is described more detailed in the following subsection.

C. Architecture Model

The System Composer model describing the ARINC 653 primitives hierarchy contains multiple levels of abstraction. The highest level describes a generic IMA architecture. The IMA architecture can contain multiple GPMs. Each GPM consists of a hypervisor which manages partitions alongside the partition schedules, and performs health monitoring of the different partitions. A more detailed description of the system is presented in [18]. For the use case, six blocks were created. The six blocks consist of four partitions for the AFAS applications, one block for the health monitoring information, and one block for the partition schedules. There are three conventional partitions, containing the TAWS, CAS, and TLS-ADSB applications. One spare partition for the TLS-CV application is added to the configuration. Each partition can have an arbitrary number of interfaces, or rather, ports. Different partitions can be connected through ports. Here, the TAWS, TLS-ADSB, and TLS-CV partitions are connected to the CAS partition using sampling ports. The complete model is shown in Fig. 4.

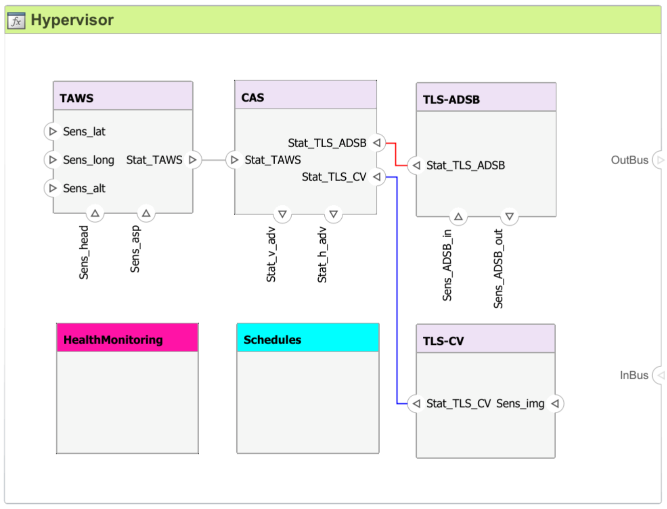

Fig. 4: System Composer model illustrating an AFAS instance

With the provided model, it is possible to model multiple GPMs in one IMA system. For each GPM, a configuration is generated using a pragmatic and concise script in MATLAB. The script parses the System Composer model and extracts information as required. The configuration, which is then saved in XML format, is used to initiate the software integration in subsequent development steps. It is important to note that a complete and valid configuration required for a dynamic reconfiguration of the avionics system needs to be specified at design time. This means that model instances are created from the meta model in System Composer and validated for the requirements set by ARINC 653. Spare partitions need to be designed in a way that they capture all necessary interfaces to substitute inter-partition connections from the partitions they replace. In this case, the CV application is used as a backup system. It captures interfaces and takes over functionality from the TLS-ADSB application.

D. Software Integration

In Simulink, the system model is generated from an ARINC 653-compliant XML configuration using tools provided by the ARINC 653 blockset. This model consists of a referenced sub-model per partition and pre-configured port blocks for inter-partition communication. Both queuing and sampling ports are supported and are linked together by name matching. Scheduling information is incorporated into the execution interval of the referenced model. By defining multiple partitions referencing the same application model, a system can be prepared for dynamic reconfiguration scenarios. The simulation ability of the blockset facilitates model development and evaluation within Simulink and is one of the advantages in this workflow. In simulation mode, the model is not compiled into target binaries but is kept within the Simulink environment. This allows for deeper insights into dataflow and model behavior. When working with the well tested and simulated model, there is no added manual coding required. This is beneficial, as manually added code could potentially introduce errors or diverging behavior. Instead, tools and automated procedures are utilized to generate the executable files for an embedded target from here.

The PikeOS Gnu Compiler Collection based toolchain is registered in Simulink Embedded Coder so it can be invoked directly from within Simulink on the generated code and build files. This integration uses template C files to automatically generate PikeOS compatible applications for the Application/Executive (APEX) personality. Together with a simple script, this allows compilation of the complete system model into multiple PikeOS application binaries from within Simulink. To adapt the separation principle of PikeOS, each partition model is compiled into its own binary application which will communicate through the configured queueing and sampling ports. This is possible because the data exchange between the models does not use Simulink signals, but the specific ARINC 653 port blocks which generate code for the respective ARINC 653 interface.

The ARINC 653-compliant XML configuration from the root IMA Model can also be used to simplify the PikeOS system configuration in CODEO. With a pragmatic script it is transformed into APEX partition components and scheduling information which can then be imported into a PikeOS integration project. By using the same naming scheme as the XML import into Simulink, no manual modifications are required to link partition ports or assign application binaries. Instead, port connections between source and destination ports in ARINC 653 partitions with the same name are automatically connected. Together with the per-partition binaries built from Simulink the PikeOS boot image is generated.

Scheduling information is also converted into PikeOS compatible configuration and does not need to be entered manually. CODEO provides a graphical visualization of the schedule. Fig. 5 shows the schedule modeled for the use case. Because the Simulink model does not include hardware specific drivers or applications, manual configuration may be required to set up additional device drivers and their connections to the APEX partitions.

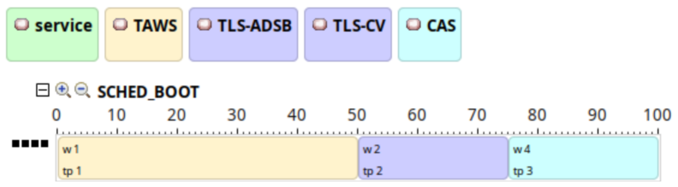

Fig. 5: CODEO visualization of imported scheduling information

E. Embedding on Target Platform

In order to bring the complete model on an embedded target, the chosen platform can be easily configured within CODEO. While the high-level model is already configured through the imported XML file, hardware specific functions may require manual adaption. This includes setting the spare partitions to IDLE startup mode, adding and configuring drivers and choosing the boot mode.

Communication channels between model partitions are added automatically. Those connecting to device drivers have to be set up by the integrator. The resulting setup after import is illustrated in Fig. 6 where some channels are automatically created while other ports stay unconnected. This is the same architecture as in the System Composer model illustrated in Fig. 4.

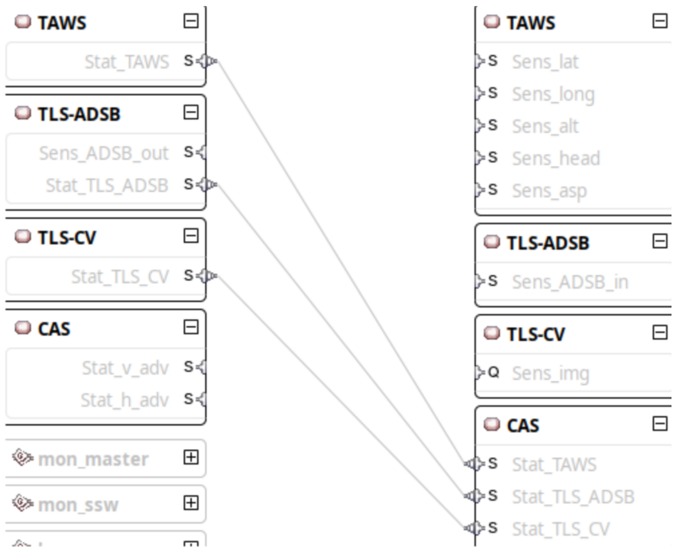

Fig. 6: CODEO visualization of partition connections.

For this demo, the QEMU emulation platform is used. The process is identical to when using a real target, though, because of the board abstraction in CODEO. From the CODEO integration project a single PikeOS boot image is generated which includes all executable partition binaries as well as the hypervisor RTOS. This PikeOS boot image is launched inside the QEMU emulation environment. Initially, the TLS-CV partition does not start. For demonstration, the PikeOS monitor connection is used to show and manipulate partitions during runtime, as shown in Listing 1:

```
PikeOS# lsp
ID Name      Type    Mode
-- ----      ----    ----
1  TAWS      PikeOS  Normal
2  TLS-ADSB  PikeOS  Normal
3  CAS       PikeOS  Normal
4  TLS-CV    PikeOS  Idle
10 service   PikeOS  Cold Start
```

Listing 1: PikeOS monitor – initial state

By executing `phalt TLS-ADSB` and `preboot TLS-CV` through the PikeOS monitor the system can be reconfigured to switch to the spare partition. In a running system, health monitoring or other external events can be used to trigger dynamic reconfiguration of the system. Listing 2 shows the initial state of the system.

```
PikeOS# lsp
ID Name      Type    Mode
-- ----      ----    ----
1  TAWS      PikeOS  Normal
2  TLS-ADSB  PikeOS  Idle
3  CAS       PikeOS  Normal
4  TLS-CV    PikeOS  Normal
10 service   PikeOS  Cold Start
```

Listing 2: PikeOS monitor – reconfigured state

The demo system periodically prints a line from each running partition to keep track of the current system status. Once the reconfiguration takes place, one can observe the switch from partition 2 to partition 4, as shown in Listing 3:

```
APEX Partition 1 starting up.
APEX Partition 2 starting up.
APEX Partition 3 starting up.
Here is Process 1 in Partition 1
Here is Process 1 in Partition 2
Here is Process 1 in Partition 3
Here is Process 1 in Partition 1
Here is Process 1 in Partition 2
Here is Process 1 in Partition 3
...
APEX Partition 4 starting up.
Here is Process 1 in Partition 3
Here is Process 1 in Partition 1
Here is Process 1 in Partition 4
Here is Process 1 in Partition 3
Here is Process 1 in Partition 1
Here is Process 1 in Partition 4
```

Listing 3: Observation of the reconfiguration from *Partition 3* to *Partition 4* during runtime

V. Discussion

The emphasis on dynamic reconfiguration within IMA modules highlights the need for robust frameworks and methodologies to capture, model, verify, and implement partition behaviors to adapt to changing operational demands. This focus on dynamic reconfiguration underscores the importance of flexibility and adaptability in avionics systems. This is especially evident in the face of failures or abnormal conditions monitored by a health system that invokes reconfiguration actions. By aligning with industry standards and leveraging custom tools for configuration generation, the development process and adherence to safety-critical requirements can be enhanced, as shown in this paper.

The exemplary implementation of an avionics configuration presented in this paper has some limitations. In practice, the applications running on an RTOS like PikeOS may be more extensive than the ones integrated here. Also, the implementation of dynamically reconfigurable partitions is minimalistic and could span much more than the few partitions shown for this exemplary implementation. Model checks can make the discussed setup more robust by only allowing valid configurations to be generated from the meta model. Constraints can help the developer adhere to the three major requirements captured by the ARINC 653 standard, that is, time isolation, space isolation, and fault coverage. Moreover, the paper does not consider multi-core systems or metrics to quantitatively evaluate the dynamic reconfiguration process. With regard to metrics, the execution times of the partitions and latencies during reconfiguration should be observed.

VI. Conclusion

The work presented in this paper illustrates a comprehensive framework for developing ARINC 653-compliant dynamically reconfigurable avionics systems. The framework encompasses various aspects of systems engineering, software integration, and end system deployment. It emphasizes a model-driven engineering approach using tools like System Composer, Simulink, and the PikeOS RTOS for the development of dynamically reconfigurable avionics systems.

One of the key takeaways of this paper is the importance of industry standards for the development of the presented avionics systems. These standards provide guidelines for the development, integration, and deployment of avionics systems, emphasizing the need for compliance with safety regulations and best practices.

Moreover, the document highlights the significance of model-driven development, modeling tools, and robust validation strategies for ensuring the integrity, compliance, and safety of avionics architectures. By leveraging domain-specific modeling languages, experts can enhance the precision and flexibility of complex avionics systems, such as dynamically reconfigurable avionics. This ultimately improves verification and validation procedures for embedded real-time software in dynamically reconfigurable and ARINC 653-compliant systems.

The systematic approach presented in this paper extends the model-driven approach to developing ARINC 653-compliant avionics systems. System parameters are captured in a high-level meta model from which configurations for deployment on target platforms are generated. This approach enhances the development process, ensures consistency across different stages of development, and facilitates the integration of software components with varying levels of criticality. The limitations of the paper mentioned before will be explored in future research.

VII. Acknowledgments

The authors thank the associates Juan Valverde and Mark McBroom from The MathWorks who provided insight and expertise that assisted the research. Their contributions in the field of dynamic reconfiguration and avionics engineering with Simulink greatly facilitated this work.

This work is part of a project that has received funding from the Federal Ministry for Economic Affairs and Climate Protection (BMWK) of Germany as part of the aviation research program LuFo VI-2 under Grant Agreement No 20Y2105 – WISDOM: Wing Integrated Systems Demonstration On Mechatronic Rig.

References

[1] Ákos Horváth and Dániel Varró. "Model-driven development of ARINC 653 configuration tables". In: *29th Digital Avionics Systems Conference*. 2010, 6.E.3-1-6.E.3–15. DOI: 10.1109/DASC.2010.5655451.

[2] Julian Schöpf, Björn Annighöfer, and Reinhard Reichel. "A Meta-Model and Transformation Schema for the Automated Generation of ICDs in an Automated Development Process of IMA System Functions". In: *7th International Workshop on Aircraft System Technologies*. Feb. 2019.

[3] Yıldız Uludağ et al. "Model-Based IMA Platform Development and Certification Ecosystem". In: *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*. Oct. 2023, pp. 1–11. DOI: 10.1109/DASC58513.2023.10311115.

[4] RTCA. *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*. Standard. Washington, D.C., USA: RTCA, Nov. 2005.

[5] ARINC. *Avionics Application Software Standard Interface*. Standard. Bowie, MD, USA: Aeronautical Radio Incorporated, Dec. 2019.

[6] NATO. *ASAAC Phase II - Final Draft of Proposed Guidelines for System Issues*. Standard. Brussels, Belgium: NORTH ATLANTIC TREATY ORGANIZATION, Nov. 2005.

[7] FAA and EASA. *Integrated modular avionics (IMA)*. Advisory Circular. Washington, D.C., USA: Federal Aviation Administration and European Union Aviation Safety Agency, Nov. 2013.

[8] ASD-STAN. *Aerospace series — Modular and Open Avionics Architectures*. Standard. Brussels, Belgium: Aerospace and Defence Industries Association of Europe - Standardization, Jan. 2010.

[9] Robert P. Goldberg. "Architectural Principles for Virtual Computer Systems". PhD thesis. Cambridge, MA: Harvard University, 1972. URL: https://apps.dtic.mil/sti/pdfs/AD0772809.pdf.

[10] Bojan Lukić et al. "State-of-the-Art Technologies for Integrated Modular Avionics and the Way Ahead". In: *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*. Oct. 2023, pp. 1–10. DOI: 10.1109/DASC58513.2023.10311229.

[11] Claire Pagetti et al. "Reconfigurable IMA platform: from safety assessment to test scenarios on the SCARLETT demonstrator". In: *Embedded Real-time Software and Systems, ERTS 2012*. Toulouse, France, Feb. 2012. URL: https://hal.science/hal-02170919.

[12] Pierre Bieber et al. "Preliminary design of future reconfigurable IMA platforms". In: *SIGBED Rev.* 6.3 (Oct. 2009). DOI: 10.1145/1851340.1851349.

[13] Guy Durrieu et al. "DREAMS about reconfiguration and adaptation in avionics". In: *ERTS 2016*. Toulouse, France, Jan. 2016. URL: https://hal.science/hal-01258701.

[14] Wanja Zaeske et al. "Towards Enabling Level 3A AI in Avionic Platforms". In: *Software Engineering 2023 Workshops*. Bonn: Gesellschaft für Informatik e.V., 2023, pp. 189–207. DOI: 10.18420/se2023-ws-18.

[15] A. Cook. "ARINC 653 — Challenges of the present and future". In: *Microprocessors and Microsystems* 19.10 (1995), pp. 575–579. ISSN: 0141-9331. DOI: https://doi.org/10.1016/0141-9331(96)84158-5.

[16] Kenneth A. Seeling. "Reconfiguration in an integrated avionics design". In: *15th DASC. AIAA/IEEE Digital Avionics Systems Conference*. 1996, pp. 471–478. DOI: 10.1109/DASC.1996.559202.

[17] Víctor López-Jaquero et al. "Supporting ARINC 653-based Dynamic Reconfiguration". In: *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. 2012, pp. 11–20. DOI: 10.1109/WICSA-ECSA.212.9.

[18] Bojan Lukić et al. "Automated Configuration of ARINC 653-Compliant Avionics Architectures". In: *AIAA SCITECH 2024 Forum*. Jan. 2024. DOI: 10.2514/6.2024-1856.

[19] FAA. *Technical Standard Order (TSO)– C151b, Terrain Awareness and Warning System*. Standard. Washington, D.C., USA: Federal Aviation Administration, Dec. 2002.

[20] FAA. *Installation of Terrain Awareness and Warning System (TAWS) Approved for Part 23 Airplanes*. Standard. Washington, D.C., USA: Federal Aviation Administration, June 2000.

[21] RTCA. *Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II)*. Standard. Washington, D.C., USA: RTCA, Mar. 2007.

[22] FAA. *Airworthiness Approval of Traffic Alert and Collision Avoidance Systems (TCAS II)*. Standard. Washington, D.C., USA: Federal Aviation Administration, Mar. 2014.

[23] Commercial Flight Standards Division (AARTF). *Automatic Dependent Surveillance - Broadcast (ADS-B) Operational and Maintenance Considerations*. Standard. Ottawa, Canada: Transport Canada, July 2021.

[24] Jasper Sprockhoff et al. "Model-Based Systems Engineering for AI-Based Systems". In: *AIAA SCITECH 2023 Forum*. Jan. 2023. DOI: 10.2514/6.2023-2587.