Automated Configuration of ARINC 653-Compliant Avionics Architectures

Bojan Lukić[*] Sven Friedrich[†] Tim Schubert[‡] Umut Durak[§]
German Aerospace Center (DLR), Braunschweig, Germany

The development of Integrated Modular Avionics (IMA) has revolutionized the avionics industry by allowing multiple functions to be present on the same General Processing Module instead of a single function per Line-Replaceable Unit in federated architectures. However, the strict restrictions and expensive verification processes associated with IMA platforms have led to the establishing of standards and guidelines to reduce development and verification costs and time. One such standard is the ARINC 653 Standard, which outlines, amongst others, the necessary data for specifying any ARINC 653 configuration. This paper proposes a model-driven approach for the automated configuration, verification, and validation of ARINC 653-compliant avionics architectures. The approach involves using System Composer and Simulink for modeling and configuring the system, as well as generating ARINC 653-compliant configurations in XML format. The paper exemplifies an expandable pattern for modeling and automatically configuring ARINC 653-compliant systems, highlighting the potential for system verification and validation at both the design and software implementation stages. The results show that Model-Driven Engineering of ARINC 653-compliant avionics architectures is a viable way to isolate parts of the engineering process and increase the automation and validity of the system.

I. Nomenclature

IMA	=	Integrated Modular Avionics
GPM	=	General Processing Module
VM	=	Virtual Machine
$APEX$	=	APplication EXecutive
API	=	Application Programming Interface
CPU	=	Central Processing Unit
DAL	=	Design Assurance Level
MDD	=	Model-Driven Development
$DSML$	=	Domain-Specific Modeling Languages
UML	=	Unified Modeling Language
$V\&V$	=	Verification & Validation
MDA	=	Model Driven Architecture
$RTOS$	=	Real-Time Operating System

II. Introduction

In the ever-evolving world of aviation, technological advancements play a vital role in enhancing safety, efficiency, and reliability. One such breakthrough is the concept of Integrated Modular Avionics (IMA), which revolutionizes the way avionics systems are designed and implemented in aircraft.

IMA consolidates multiple avionics functions onto a common hardware platform. Before the introduction of IMA, aircraft systems were predominantly developed using a federated architecture, where each system had its own dedicated hardware. However, IMA takes a different approach by providing a shared computing platform that hosts multiple applications, reducing weight, power consumption, and overall system complexity [1].

[*]Research Scientist, Institute of Flight Systems.
[†]Research Scientist, Institute of Flight Systems.
[‡]Research Scientist, Institute of Flight Systems.
[§]Group Leader, Institute of Flight Systems, AIAA Associate Fellow.

At its heart, IMA incorporates the ARINC 653 standard, which is defined by Aeronautical Radio, Inc. (ARINC) and involves the requirements for partitioning and scheduling of software applications on an IMA platform. ARINC 653 ensures that different software modules, known as partitions, coexist in a safe and independent environment, preventing interference and ensuring fault tolerance. By adhering to this standard, aircraft manufacturers can achieve a high-level of system reliability and maintainability.

The Model-Driven Development (MDD) of avionics architectures has emerged as a powerful paradigm to design and implement IMA systems [2–5]. This approach utilizes models to describe the behavior and structure of avionics systems, allowing for rapid prototyping, verification, and validation of complex systems. Further, MDD enables engineers to simulate and analyze the system's performance, optimize resource allocation, and ensure compliance with safety regulations.

By employing a model-driven approach, engineers can create high-level models representing the functionality of the system, interactions, and timing requirements. These models serve as a blueprint for generating ARINC 653-compliant configurations, ensuring that the final software implementation adheres to the standard's strict guidelines. The model-driven configuration of ARINC 653-compliant partitioned avionics architectures involves several steps, including system modeling, software architecture design for partitions, and Verification & Validation (V&V). Through these steps, engineers can effectively manage the complexity of IMA systems, allocate computing resources to different software components, define their execution schedules, and verify the system's behavior against its requirements.

This paper explores the MDD of ARINC 653-compliant avionics architectures. This approach enables several beneficial development processes: 1. The use of models as a means for single source of truth development, 2. Generic ARINC 653-compliant profile creation with metamodels, 3. Automatic generation of configurations from profile instances, and 4. V&V of these architectures. The remaining paper is structured as follows: In Section III, background information on technologies and methodologies is presented. Section IV presents the current state in configuration generation for ARINC 653-compliant systems. The main part, Section V, discusses MDD for automatically generating said configurations and possible V&V methods for such models. Results are discussed in Section VI, before concluding the paper with Section VII.

III. Background

IMA contains many function on the same General Processing Module (GPM) as opposed to a single function per Line-Replaceable Unit (LRU) in federated architectures [1]. As restrictions are strict and verification is expensive, a few standards established themselves in the avionics industry aiming to reduce development and verification cost and time of IMA platforms such as ARINC 653 and ED-12C/DO-178C.

A guideline for certification of safety-critical software in airborne systems is the document ED-12C/DO-178C [6]. It defines Design Assurance Levels (DALs) A, B, C, and D, which classify software based on their level of criticality and required rigor for development and verification. In the context of ED-12C/DO-178C, the development of state-of-the-art IMA refers to the deployment of functions of mixed DAL (A to D) inside the same GPM. Hypervisors developed according to ARINC 653 are already being used in IMA contexts, with some safety-critical functions reaching up to the highest of DAL, i.e., DAL A.

A. ARINC 653

Focusing on the execution of software on GPMs, the ARINC 653 Standard proposes the APplication EXecutive (APEX). APEX is a set of behaviors and Application Programming Interface (API) functions to be implemented by avionic software vendors into their hypervisor. It implements the isolation of software running on the GPM and prevents erroneous software components from affecting others according to ED-12C/DO-178C.

Relevant for this paper is the APEX isolation primitive *Partition* as well as its scheduling and communication channels.

Isolation is enforced in the unit of partitions with each having its exclusive memory region to execute upon. Not only does this include the disk memory but also the Random-Access Memory (RAM). As for the scheduling, a static schedule is carried out, enforcing preset time windows to be used by each partition [7]. For the configuration of the static schedule, the most important datum is the *major frame*, that determines the smallest frequency at which partition time windows can be configured. This arrangement is then done inside of the major frame, periodically repeating it as demonstrated with three partition time windows in Figure 1.

Just like with the major frame, the partitions need to be aligned periodically. This means that only three values are necessary for the configuration of each partition's time windows – *duration*, *offset*, and *period*. The duration specifies

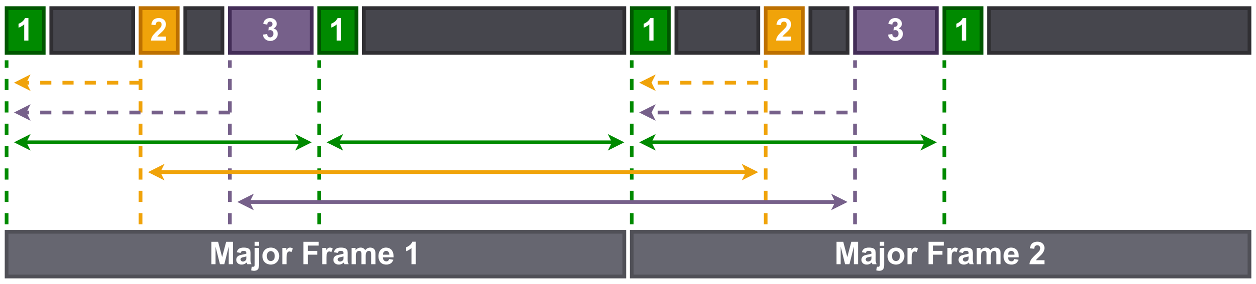

Fig. 1 Static scheduling in ARINC 653
Dashed arrows: offset. Normal arrows: period

the length for each time window of a partition and the period sets its frequency, which always needs to be a divisor of the configured major frame duration. The offset shifts the periodical occurrence of a partition's time window.

In this example, the periods of partitions 2 and 3 are equal to the major frame duration, and the period of partition 1 is half the duration of the major frame. There is no offset for partition 1, but for partitions 2 and 3. Considering an exemplary major frame duration of 1 s, the period for partition 1 is 500 ms and 1 s for partitions 2 and 3. The offset for partitions 2 and 3 would then be ~200 ms and ~350 ms, respectively.

APEX allows inter-partition communication through predefined channels, with ARINC 653 introducing two different types of channels, namely *sampling* and *queuing ports*. Both of them are unidirectional and sender and receiver can not be changed during run-time.

Sampling ports are channels with a one-to-many communication act where the last written value is retained. Because of this, sampling ports are well suited for sensor data values which are continuously updated and where missed reads do not pose an issue. Queuing ports on the other hand only support communication from one partition to another. Here, written messages are stored in a queue and, except for when the queue is full, no message is lost.

B. ARINC 653-Specific Configuration

According to ARINC 653, configurations are used for general partition specifications including inter-partition communication, partition scheduling, and health monitoring at boot time. In the configuration specifications, the root element *MODULE* contains three sequence elements: *Partitions*, *Schedules*, and *HealthMonitoring*.

The *Partitions* element describes the partitions configuration in the module and contains the sub-element *Partition*. The sub-elements of *Partition* are *PartitionDefinition* – the name and ID of the partition, *PartitionPeriodicity* – the partition periodicity, *MemoryRegions* – the memory region mapped into the partition, and *PartitionPorts* – the ports of the partition. The *Schedules* element describes the schedule configuration and contains the sub-element *PartitionTimeWindow*. The *HealthMonitoring* element represents the health-monitoring configuration for the module. It contains the sub-elements *SystemErrors* – the list of system errors, *ModuleHM* – the module level HM table, *MultiPartitionHM* – the partition level HM table for multiple partitions, and *PartitionHM* – the partition level HM table [7]. These configuration elements contain unique attributes which are not further detailed in this section. They are presented and applied in the main part of this paper. Some attributes which are, according to ARINC 653, explicitly not included in the configuration are following:

- Number of associated processor cores – the number of logical processor cores used to schedule processes associated with a partition
- Operating mode – the partition's execution state
- Partition processes – the executable processes running within partitions

These attributes are associated with the software integration part of the Systems Engineering framework. This means that the specification of these attributes is bound to the development with the respective software development tools. The division between system design and system integration within the Systems Engineering framework is illustrated in detail in the main part of this paper.

C. The ARINC 653 Configuration Data Format

The ARINC 653 standard presents an expandable XML schema that outlines the format of the necessary data for specifying any ARINC 653 configuration. This XML schema allows for defining configuration tables and elements within ARINC 653 systems. Although the standard itself does not explicitly mention it as a configuration format, XML configuration files play a role in defining the configuration structure for time and space partitions within ARINC 653-compliant systems [7].

These configuration files adhere to specific XML schemas that define the necessary configuration requirements. They lay the foundation for "an intermediate form of the configuration definition that allows the system integrator to create configuration specifications in a form that can be readily converted into an implementation-specific configuration" [7]. The schema ensures the accuracy and uniformity of the configuration data within ARINC 653 systems. Access to these XML configuration files is typically restricted to the operating system of the ARINC 653 system [8, 9].

The ARINC 653 standard does not provide detailed specifications regarding the structure or content of the XML configuration files. The exact format and elements within these files may vary depending on the specific implementation or configuration tools employed for ARINC 653 systems [10]. While the ARINC 653 standard does not explicitly discuss XML configuration, it appears that XML configuration files are commonly employed in defining the configuration data for time and space partitions in ARINC 653 systems [11, 12]. For this paper, the XML format is used to provide all necessary information for custom configuration of end systems according to ARINC 653.

D. Metamodeling with Profiles and Stereotypes

Metamodeling is the process of creating models that describe the structure and behavior of other models. In the context of MDD, metamodeling is the process of defining the syntax and semantics of modeling languages, which are then used to create models of software systems. There are two distinct forms of metamodeling: linguistic and ontological, these two forms give rise to two distinct forms of instantiations [13, 14].

Metamodeling enables the creation of Domain-Specific Modeling Languages (DSML) and provides a formal representation of the concepts and relationships within a particular domain. By defining a metamodel, a common vocabulary and set of rules for creating models can be established. These models then accurately represent the desired system or application [14]. Metamodeling in MDD involves the following key concepts:

Metamodel
A metamodel is a model that defines the structure, constraints, and semantics of other models within a specific domain. It specifies the types of elements, their relationships, and the rules that govern their usage. Metamodels are typically represented using standardized modeling languages [15].

Model
A model is an instance of a metamodel. It represents a specific system or application within a given domain. Models are created by conforming to the structure and constraints defined by the metamodel. They capture the essential aspects of the system being developed, such as its structure, behavior, and data [16].

Model Transformation
Model transformations are operations that convert models from one representation to another. They allow for the manipulation, refinement, and generation of models based on predefined rules and mappings. Model transformations in MDD enable the automatic generation of code, documentation, and other artifacts from models [17]. There are three basic types of transformations, namely model-to-model, model-to-text, and text-to-text transformation.

Model-to-model transformation refers to the process of converting one model into another model. This transformation is typically performed to bridge the gap between different modeling languages, tools, or representations, allowing for interoperability and seamless integration between different parts of a software system.

Model-to-text transformation, also known as model-to-code transformation, involves generating executable code or other textual artifacts from a high-level model. The transformation process involves defining templates or patterns that specify how elements and relationships in the source model should be mapped to code or textual representations.

Text-to-text transformation, also known as text-to-text generation, refers to the process of transforming one textual representation into another [18, 19].

Through metamodeling models that capture the essential characteristics of a system and its components can be created. Metamodeling also promotes Model-Driven engineering practices, where models serve as the primary artifacts for system development and analysis. This provides a structured approach to system development by defining the vocabulary, structure, and constraints of models within a specific domain. It promotes reusability, consistency, and automation in the development process, leading to increased productivity and improved software quality [20].

The Object Management Group (OMG) defines a standard called (MOF) for a metamodel architecture in MDD. The four-layered architecture is the Model Driven Architecture (MDA) for creating and manipulating models and metamodels. Figure 2 shows three of the four modeling layers from the MDA. The information layer consists of the particular runtime data that the modeler intends to depict [21].

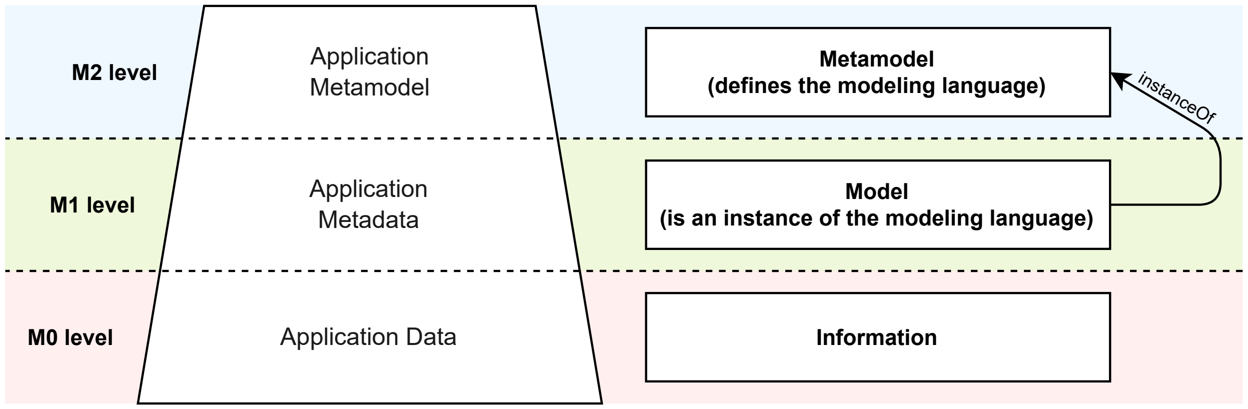

Fig. 2 Metamodeling layers and patterns in MDD

The M3 Metametamodel level is omitted. This paper focuses on the use of custom profiles as metamodels extending the SysML M2 metamodel to the M0 information level.

In the context of metamodeling, profiles and stereotypes are concepts used to extend and customize existing metamodels, such as the Unified Modeling Language (UML) metamodel. Although they complement each other, there are some differences:

Profiles
UML profiles are a method of expanding the UML to create a modeling language that is customized to a specific platform or application domain. They offer a precise way of using UML in a specific context and can be combined within the MDA context to define a series of model transformations. Private organizations and software companies can define UML profiles, and the OMG has standardized several UML profiles [22–24].

Stereotypes
Stereotypes are defined within a profile and serve as a means to extend existing metaclasses from the base metamodel. A stereotype defines new properties, operations, and constraints that can be applied to instances of the metaclass it extends. It allows users to add domain-specific characteristics and semantics to existing metamodel elements. Stereotypes that are associated with model elements are typically highlighted within the model, indicating their specialized meaning or behavior [23, 24].

By using profiles and stereotypes, metamodels can be customized to represent specific domains, industries, or application contexts. They enable the creation of DSMLs that capture the unique aspects and requirements of a particular problem domain [25, 26].

IV. Related Work

There are several publications on the development of ARINC 653 systems. Some of them are more general papers about requirements for successful IMA development, while some describe custom tools for generating configurations with transformation tools.

Uludağ et al. [27] discuss the safety, security, and certification challenges of developing a highly complex distributed IMA platform. The main challenges of the development include the necessity for the management of modular and re-usable certification processes, IMA systems with reconfiguration capability, and the safe management of shared resources within IMA. The authors propose a Model-Based System Engineering approach to address these challenges and present a practical implementation framework for the Turkish fighter program. The outcome of the paper is that the approach and framework can effectively address the challenges of developing a complex distributed IMA Platform and can be used as a basis for future avionics development projects.

Lafoz et al. [8] present the design and development of ARINC 653 systems, which are used in IMA architectures. The paper proposes a UML extension to deal with the design of these systems, and describe the use of XML schema for defining their configuration. They also present an automatic generation of configuration tables and a qualified validation process of them, as well as automatic generation of ARINC 653 artifacts in terms of code and partition tests and stubs. In particular, the paper proposes a UML approach for defining configurations with an XML schema, the automatic generation of configuration tables, and ARINC 653 artifacts.

Horváth et al. [28] show the application of MDD to the development of ARINC 653 configuration tables in the aeronautic domain in their paper. The paper presents a tool chain that generates configuration tables from high-level architecture models, and they demonstrate the benefits of using MDD in this context. The paper points out that MDD can be effectively applied to the systematic development of ARINC 653 configuration tables, and that it can improve the efficiency and quality of the development process. Additionally, the paper highlights the importance of end-to-end traceability information to support certification for V&V activities.

The paper [9] by Duprat et al. is about the model-based approach in configuration data management for LVCUGEN Flight Software. The paper presents how this approach has been implemented to ensure the consistency and qualification of each component in the software. It covers the technical context of flight software with the CNES solution LVCUGEN, the model engineering chain, and the qualification strategy for the checker tool and generator. The relevant outcomes in this paper are that the model-based approach is an effective way to manage configuration data in generic software frameworks and that it can ensure the consistency and qualification of each component in the software. The approach covers model verification and code generation from the model and integrates a qualification strategy. The methodological solution is adapted to a collaborative organization where the product is developed and integrated by increments. The paper also highlights the importance of qualification issues and proposes qualification strategies for multidomain engineering tools.

Other publications in this context are [29], which discusses a verification method of end-to-end real-time properties on IMA systems; [30], about the use of the Simulink environment to implement IMA partitions models through an ARINC 653 block set; and [11], presenting a software tool for integrating configuration data of ARINC 653 operating systems. The contribution of the paper at hand is the exemplification of an expandable pattern for modeling and automatically configuring ARINC 653-compliant systems. It involves a split between Systems Engineering and software integration with automated configuration generation and potential for validation of the system both on the design and the software implementation side.

V. ARINC 653 System Modeling, Model Transformation, and Configuration

The setup involves two parts. First, the high and medium-level description of the ARINC 653 system with its components and attributes. Second, the functional description of the system. The first part is fully modeled in System Composer, while the second part is modeled in Simulink. For the functional part in Simulink, a complete system model called *ARINC 653 blockset*[*] is provided as an add-on. This add-on is used for functional architecture modeling of ARINC 653 systems. For the functional part, other software tools are also available, as will be described shortly.

[*]Arinc 653 block set, as presented in: G. Corraro, E. Bove, L. Garbarino and E. Memoli, "A novel approach for the development and coding of avionics functionalities for IMA architectures," 2018, doi: `https://doi.org/10.1109/DASC.2018.8569824`

The setup has multiple facets: One part is concerned with modeling a discrete ARINC 653 compliant system and generating a configuration in XML format. Another part handles the integration of the said high-level system representation in SIMULINK and the automated creation of a functional model. At last, the generated functional model is used as a wrapper, i.e., a template, which can be completed and extended with code and elements. The workflow is illustrated in Figure 3.

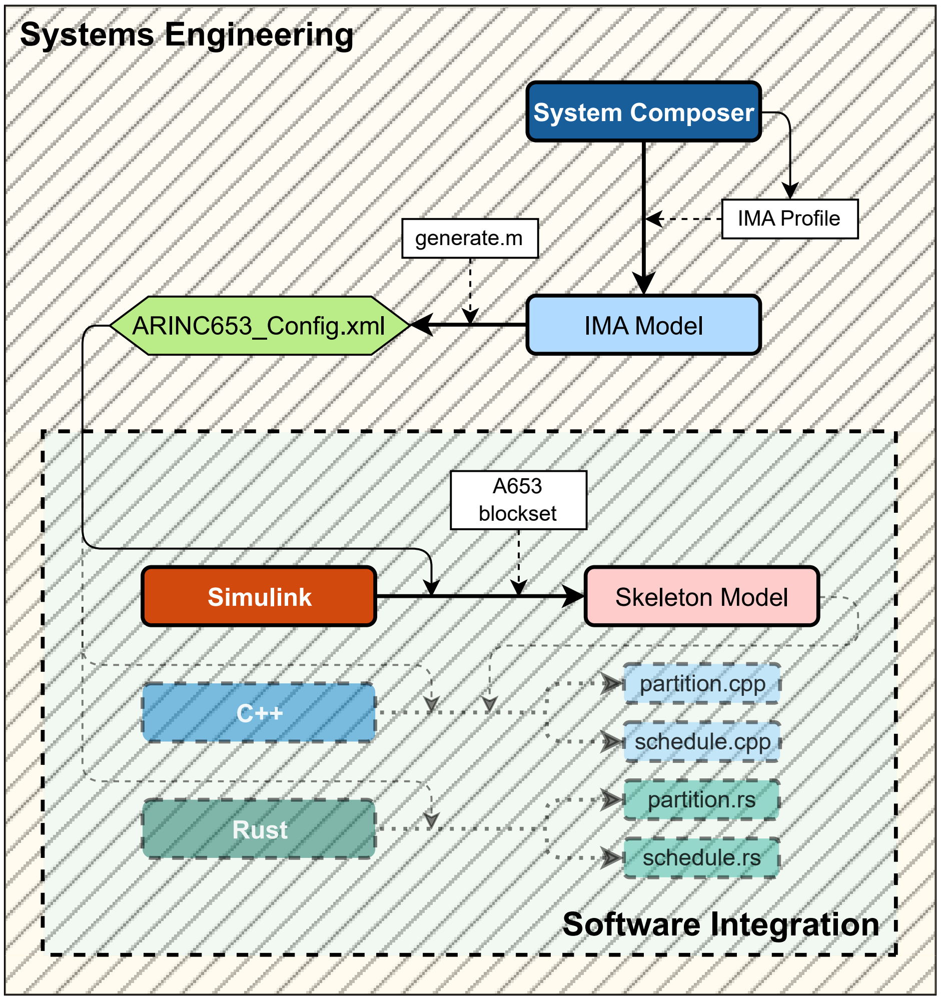

Fig. 3 Workflow followed in this paper portrayed in the systems and software engineering landscape

Apart from the software integration in Simulink, other possible platforms for system integration are illustrated in the workflow. Examples are C++ and Rust. As Simulink provides code exports in C++, the integration of the high-level representations of an ARINC 653-compliant system can further be linked to C++ software development steps.

Ultimately, the aim is the automated and seamless integration between systems design and software integration within the Systems Engineering framework. For the specific use case, this implies the automated generation of configurations from a System Composer profile and their integration in a functional Simulink model. A sample configuration taken from the ARINC 653 standard is shown in Listing 1.

A. Metamodeling of Partitions

The model contains multiple architectural layers. In the context of ARINC 653, all components making up an IMA architecture can be regarded as the highest level of abstraction. Inside multiple standardized IMA specific components can be found, such as network switches, legacy units, and the GPM. The IMA layout is illustrated in Figure 4.

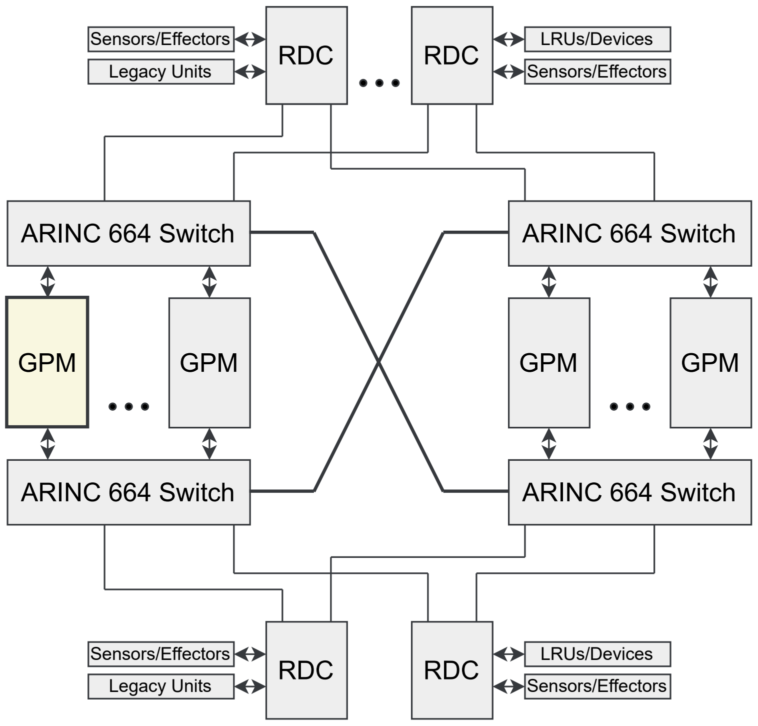

Fig. 4 Integrated System Architecture

GPMs refer to computing modules that execute application software. These modules consist of a Central Processing Unit (CPU), a Real-Time Operating System (RTOS), an allocated amount of memory, and a hypervisor. An exemplary GPM with its components is shown in Figure 5.

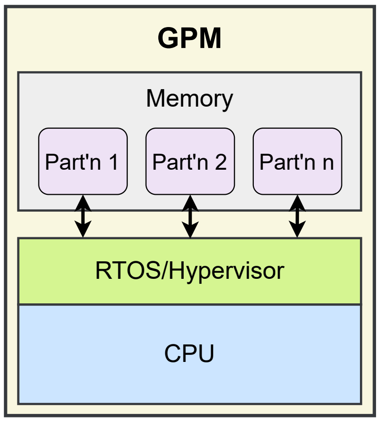

Fig. 5 One GPM with three partitions on discrete memory spaces

The CPU computes various processes, such as the flight control or flight entertainment systems. A hypervisor is software that facilitates the creation and operation of Virtual Machines (VMs), enabling a single host to support multiple VMs concurrently, each with its own operating system and set of applications. By means of virtualization, partitions can be established to host and execute applications. The number of tasks executed within a partition may vary depending on the complexity of the system. A discrete space on the memory is assigned to each partition, assuring spatial separation between partitions [31]. Given the scope of this paper specifically targeting the configuration of partitions and their schedules, only the GPM inside of an overall IMA architecture is discussed.

The avionics architecture modeled in System Composer has three levels of abstraction. The highest level contains the GPM inside a generic IMA architecture. The second level contains the hypervisor inside the GPM and the lowest level the case-specific partitions run by the hypervisor. The respective model can be seen in Figure 6.

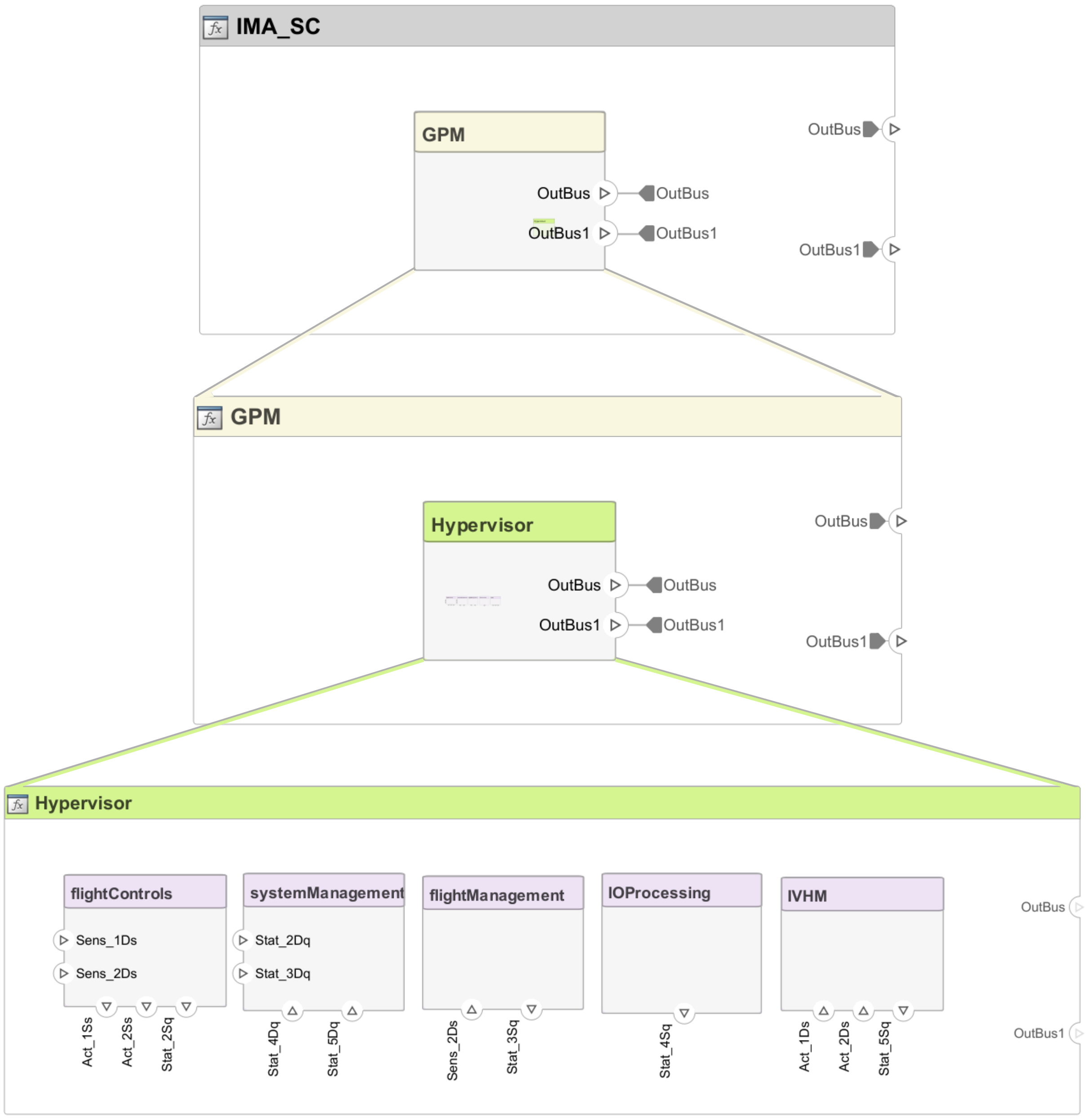

Fig. 6 Three levels of abstraction in System Composer model

B. Automated XML Configuration Generation

The profile affiliated with the metamodel contains multiple stereotypes that are part of the IMA system. These are namely *CPU*, *Hypervisor*, *Memory*, *Partition*, *Queuing Port*, *and Sampling Port*. Although each stereotype is part of the complete system, which is needed in case of extensions, the *Partition*, *Queuing Port*, and *Sampling Port* stereotypes are of interest. Each stereotype contains specific properties. The properties of *Partition* are listed in Table 1.

Property name	Type	Unit	Default
partitionDuration	unit16	ms	100
id	string		’Default_ID’
periodical	boolean		true
startTime	uint16	ms	0
periodicity	uint16	ms	500
priority	uint16		0
name	string		’partition’
memoryRam	uint32	Bytes	1048576
memoryFlash	uint32	Bytes	524288
memoryIO	uint32	Bytes	524288
accessRightsRAM	enumeration		READ_ONLY
accessRightsFlash	enumeration		READ_ONLY
accessRightsIO	enumeration		READ_ONLY

Table 1 Properties contained in the *Partition* stereotype.

The properties of *Queuing* and *Sampling Port* are listed in Table 2. Note that the property *maxNbMessage*, which is the maximum number of messages sent, is exclusive to the stereotype *Queuing Port*.

Property name	Type	Unit	Default
maxMessageSize	unit16	Bytes	4096
minMessageSize	unit16	Bytes	16
name	string		’Stat’
direction	enumeration		SOURCE
maxNbMessage	uint16	Bytes	4096

Table 2 Properties contained in the *Queuing Port* and *Sampling Port* stereotypes.

The properties from the stereotype *Profile* contain the content from the sample XML configuration of the ARINC 653 standard, partly shown in Listing 1, and more. The profile is generic and scales with additions and modifications to existing reference configurations. Model transformation refers to the transformation of models to their representation in a different format. This transformation can be accomplished in different forms, with one example being model-to-text transformation. For the automated generation of configurations from the System Composer model, model-to-text transformations are implemented pragmatically using the System Composer API with MATLAB scripts. The script, wrapped in a function, parses and writes model properties from the System Composer model to an XML configuration file. It obeys the logic shown in the pseudocode in Table 3.

This logic can be applied to any modeling software with its respective model transformation language, i.e., model transformation framework. The complete script is illustrated in Listing 2. Part of the code for writing properties to a file is omitted for reasons of conciseness.

C. ARINC 653 Block Set

The ARINC 653 block set is a set of blocks used to implement IMA partitions models in the Simulink environment, as part of the methodology and tool set developed for rapid prototyping of avionics functionalities on IMA architectures. The block set includes blocks for partition management, process management, inter-partition communication, and time

Input: System Composer *model*.

```
traverse model to Hypervisor architecture level
for each Object in Hypervisor
    if Object is of stereotype Partition
        for each property in Object
            save property to property_array
        end

        for each Port in Object
            for each property in Port
                buffer property to temp_array
            end
            save temp_array to port_array
        end

        build XML file with information from property_array
        and port_array
    end
end
```

Output: XML configuration file; **optional**: *property_array*, *port_array*

Table 3 Pseudo code for parsing and writing model properties from System Composer to an XML configuration file.

management. These blocks are used to model the functional IMA partition infrastructure, which is the set of services provided by the ARINC 653 RTOS to support the execution of multiple applications on a single hardware platform. The ARINC 653 block set allows for the functional modeling of the partition infrastructure in a high-level environment, which facilitates the development and testing of avionics functionalities on IMA architectures. The block set is integrated with the Simulink infrastructure partition model, which is given as input to the customized automatic code generator environment in order to generate compliant VxWorks 653[†] code ready to be deployed on IMA system. The use of a single code generation environment for the automatic code generation of whole partitions (partition infrastructure and applications) allows for the need to follow a single qualifying process in order to develop safety-critical systems.

Further, the blockset defines mandatory APIs that an ARINC653 OS must support. The APIs are broken into 6 categories with a total of 57 services, namely Partition Management – 2 services, Process Management – 14 services, Time Management – 4 services, Inter-Partition Communication – 11 services, Intra-Partition Communication – 22 services, Health Monitoring – 4 services, and an XML schema for specifying an ARINC 653 configuration. The behavior specified in ARINC standard 653 is implemented with Simulink blocks. These blocks are used to generate functional code for ARINC 653 services. The outputs of the architectural design phase described in the previous subsections are inputs to the partitions development process in Simulink environment. More specifically, the XML configuration file generated from the System Composer model is used as an artifact in the block set, as shown in Figure 3. The developed toolbox allows modeling an IMA partition in terms of APEX blocks for partition and process management, intra- and inter-partition communication, time management, and health monitoring. The Simulink blocks of the ARINC 653 package are implemented as fully in-lined S-Functions based on ARINC 653 standard APEX and ARINC 653 RTOS specific code execution characteristics. The S-Function of the software application and the related Target Language Compiler (TLC) file are generated through the standard MATLAB Embedded Coder and Legacy Code

[†]Wind River VxWorks 653, as presented in: Wind River, "VxWorks 653 Multi-core Edition," 2022, url: `https://www.windriver.com/resource/vxworks-653-product-overview`

Tool application. Such outputs are integrated in the Simulink infrastructure partition model implemented through the ARINC 653 block set.

Currently, the block set does not provide any automated way of generating models from the XML reference file. The functional blocks are created manually and cross-checked with the deposited configuration file. Part of this paper is aimed at developing a fully integrated development scheme for ARINC 653-compliant avionics architectures. For this reason, the automatic software integration and development of those avionics architectures in Simulink and other software development tools is not in the scope of this paper.

D. Model Correctness

In order for the ARINC 653 configurations to be valid when implemented in end systems, model correctness has to be guaranteed at design time. In the scope of this paper, the correctness of a model can be determined with V&V, as well as with simulation & testing.

Verification refers to the process of evaluating a system model or design to determine if it meets the specified requirements and adheres to defined standards. It involves checking the consistency, completeness, and correctness of the system model or design. Verification activities focus on ensuring that the system is built right and that it satisfies the intended functionality. Validation, on the other hand, is the process of evaluating a system model or design to determine if it meets the needs and expectations of the stakeholders. It involves assessing the system's performance, functionality, and suitability for its intended use. Validation activities focus on ensuring that the right system is built and that it satisfies the stakeholders' requirements [32].

Simulation involves creating a virtual representation of a system or process to analyze its behavior and performance. It allows engineers to study and evaluate the system's response under different conditions and scenarios without the need for physical prototypes. Simulation enables the exploration of system behavior, identification of potential issues, and optimization of system design. It aids in understanding system dynamics, validating system requirements, and making informed decisions during the development process. Testing involves executing the system model or design to assess its functionality, performance, and compliance with requirements. It aims to identify defects, errors, or deviations from expected behavior. Testing activities include designing test cases, executing tests, and analyzing the results. Testing helps to ensure that the system functions as intended and meets the stakeholders' requirements. It also helps to identify and rectify any issues before the system is deployed or implemented [33].

Taking Figure 3 as a reference, V&V is performed in the design phase, while simulation & testing is exclusive to the software integration phase. This is evident when considering the different levels of abstractions these two phases cover. Delange et. al. [34] describe a validation approach for ARINC 653 avionics architectures. The authors discuss four steps for checking the correctness of generated ARINC 653 configurations, three of which are of interest in this paper:

1) Time isolation – each partition is executed at least one time during each scheduling period and; The consistency of the major time frame according to partitions time frames
2) Space isolation – association of each memory segment with a single partition
3) Fault coverage – recovering faults on the module, partition, and process layer

These properties can be checked in different ways. Delange et. al. [34] us a custom Architecture Analysis & Design Language (AADL). For the paper at hand, more methods, such as Object Constrain Language (OCL) (see [35]), for model correctness will be investigated in the future. Moreover, runtime analysis will be conducted with the software integration on end systems, involving simulation & testing.

VI. Discussion

The main methods used in this paper are MDD, generic profile creation, and automated configuration generation. These methods have significant implications for the development and verification of ARINC 653-compliant avionics architectures.

MDD is used as a means for single source of truth development. This approach ensures the validity of the integrated system during design time. The use of generic profiles facilitates the creation of various architecture instances with a single source of truth, before continuing with software integration.

Automated configuration generation can be leveraged in this approach by using generic model transformation. This approach can reduce the time and effort required for configuration generation, which is particularly important given the strict restrictions and expensive verification processes associated with IMA platforms.

The implication of these methods is that the MDD of ARINC 653-compliant avionics architectures is a complex process that requires careful consideration of various technologies and methodologies. However, the use of these methods can reduce development and verification time of IMA platforms.

VII. Conclusion

The MDD of ARINC 653-compliant avionics architectures is a complex process that requires careful consideration of various technologies and methodologies. This paper provides a comprehensive overview of this process, highlighting the benefits of using a MDD approach for the development of IMA platforms.

One of the key benefits of this approach is the use of MDD as a means for single source of truth development. This allows for the creation of generic ARINC 653-compliant profiles from metamodels, which can then be used to automatically generate configurations from profile instances. This approach can reduce development and verification costs and time of IMA platforms, which is particularly important given the strict restrictions and expensive verification processes associated with these platforms.

In conclusion, this paper provides a clear and concise overview of the MDD of ARINC 653-compliant avionics architectures, highlighting the benefits of this approach in terms of rapid prototyping, verification, and validation of complex systems. The use of generic profiles and automated configuration generation can further facilitate the creation of various architecture instances with a single source of truth. Overall, this paper provides valuable insights into the development and verification of IMA platforms, specifically focusing on the ARINC 653 Standard.

References

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

[2] Annighöfer, B., "Model-driven Development and Simulation of Integrated Modular Avionics (IMA) Architectures," *SNE Simulation Notes Europe*, Vol. 28, 2018, pp. 61–66. https://doi.org/10.11128/sne.28.tn.10414.

[3] Annighoefer, B., Brunner, M., Schoepf, J., Luettig, B., Merckling, M., and Mueller, P., "Holistic IMA Platform Configuration using Web-technologies and a Domain-specific Model Query Language," *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pp. 1–10. https://doi.org/10.1109/DASC50938.2020.9256726.

[4] Halle, M., and Thielecke, F., "Next generation IMA configuration engineering - from architecture to application," *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 2015, pp. 6B2–1–6B2–13. https://doi.org/10.1109/DASC.2015.7311445.

[5] Schöpf, J., Annighöfer, B., and Reichel, R., "A Meta-Model and Transformation Schema for the Automated Generation of ICDs in an Automated Development Process of IMA System Functions," *7th International Workshop on Aircraft System Technologies*, 2019.

[6] RTCA and EUROCAE, "SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION," Standard, Radio Technical Commission for Aeronautics and European Organization for Civil Aviation Equipment, Malakoff, France, Jan. 2012.

[7] ARINC, "Avionics Application Software Standard Interface Part 1 Required Services," Standard, Aeronautical Radio Incorporated, Bowie, MD, USA, Dec. 2019.

[8] Lafoz, I., Mozas, M. A., Charrier, O., and Fernández de La Hoz, C., "IMA Systems Development and Configuration: From UML to Binary. A Proposal of UML Profile to be used with the ARINC 653 Standard." *Embedded Real Time Software and Systems (ERTS2008)*, toulouse, France, 2008. URL https://insu.hal.science/insu-02270111.

[9] Duprat, S., Haugommard, A., Galizzi, J., Navarro, C., and Masmano, M., "Model based approach in configuration data management for LVCUGEN Flight Software," *ESA MBSE2021 - Model Based Space Systems and Software Engineering*, 2021. URL https://indico.esa.int/event/386/contributions/6235/.

[10] Wilson, A., and Preyssler, T., "Incremental certification and Integrated Modular Avionics," *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, 2008, pp. 1.E.3–1–1.E.3–8. https://doi.org/10.1109/DASC.2008.4702768.

[11] Paeng, B.-J., Ha, O.-K., and Jun, Y.-K., "Software Tool for Integrating Configuration Data of ARINC 653 Operating Systems," *2015 8th International Conference on Grid and Distributed Computing (GDC)*, 2015, pp. 20–23. https://doi.org/10.1109/GDC.2015.17.

[12] teum Choi, E., Ha, O.-K., and Jun, Y.-K., "Configuration Tool for ARINC 653 Operating Systems," *International Conference on Multimedia and Ubiquitous Engineering*, 2014. URL https://api.semanticscholar.org/CorpusID:61974431.

[13] Atkinson, C., and Kuhne, T., "Model-driven development: a metamodeling foundation," *IEEE Software*, Vol. 20, No. 5, 2003, pp. 36–41. https://doi.org/10.1109/MS.2003.1231149.

[14] Siegel, J., "Object Management Group Model Driven Architecture (MDA) MDA Guide rev. 2.0," Tech. rep., Object Management Group, 2014. Available at http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.

[15] Favre, J.-M., and Nguyen, T. T. T., "Towards a Megamodel to Model Software Evolution Through Transformations," *SETra@ICGT*, 2005. URL https://api.semanticscholar.org/CorpusID:6174320.

[16] Bézivin, J., and Gerbé, O., "Towards a precise definition of the OMG/MDA framework," *Automated Software Engineering*, 2001, pp. 273– 280. https://doi.org/10.1109/ASE.2001.989813.

[17] Mian, Z., Bottaci, L., Papadopoulos, Y., Sharvia, S., and Mahmud, N., "Model Transformation for Multi-objective Architecture Optimisation of Dependable Systems," *Dependability Problems of Complex Information Systems*, edited by W. Zamojski and J. Sugier, Springer International Publishing, Cham, 2015, pp. 91–110.

[18] Neto, V. V. G., "A simulation-driven model-based approach for designing software intensive systems-of-systems architectures," Phd thesis, Université de Bretagne Sud; Universidade de São Paulo (Brésil), June 2018. Available at https://theses.hal.science/tel-02146340v1/document.

[19] Mens, T., and Van Gorp, P., “A Taxonomy of Model Transformation,” *Electronic Notes in Theoretical Computer Science*, Vol. 152, 2006, pp. 125–142. https://doi.org/https://doi.org/10.1016/j.entcs.2005.10.021, URL https://www.sciencedirect.com/science/article/pii/S1571066106001435, proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005).

[20] Sauer, S., *Applying Meta-Modeling for the Definition of Model-Driven Development Methods of Advanced User Interfaces*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 67–86. https://doi.org/10.1007/978-3-642-14562-9_4, URL https://doi.org/10.1007/978-3-642-14562-9_4.

[21] Cetinkaya, D., and Verbraeck, A., “Metamodeling and model transformations in modeling and simulation,” *Proceedings - Winter Simulation Conference*, 2011, pp. 3043–3053. https://doi.org/10.1109/WSC.2011.6148005.

[22] Fuentes, L., and Vallecillo, A., “An introduction to UML profiles,” *UPGRADE, The European Journal for the Informatics Professional*, Vol. 5, 2004.

[23] Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed., Addison-Wesley, Boston, MA, 2003.

[24] Selic, B., “UML 2.0 Superstructure Specification,” Tech. rep., Object Management Group, 2005. Available at http://www.omg.org/cgi-bin/doc?ptc/2004-10-02.

[25] Giachetti, G., Marín, B., and Pastor, Ó., “Integration of domain-specific modelling languages and UML through UML profile extension mechanism,” *Int. J. Comput. Sci. Appl.*, Vol. 6, 2009, pp. 145–174. URL https://api.semanticscholar.org/CorpusID:7952602.

[26] Abouzahra, A., Bézivin, J., Fabro, M. D. D., and Jouault, F., “A Practical Approach to Bridging Domain Specific Languages with UML profiles,” , 2005. URL https://api.semanticscholar.org/CorpusID:5620908.

[27] Uludağ, Y., Bayoğlu, O., Candan, B., and Yılmaz, H., “Model-Based IMA Platform Development and Certification Ecosystem,” *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, not yet published.

[28] Horváth, A., and Varró, D., “Model-driven development of ARINC 653 configuration tables,” *29th Digital Avionics Systems Conference*, 2010, pp. 6.E.3–1–6.E.3–15. https://doi.org/10.1109/DASC.2010.5655451.

[29] Lauer, M., Ermont, J., Boniol, F., and Pagetti, C., “Latency and freshness analysis on IMA systems,” *ETFA2011*, 2011, pp. 1–8. https://doi.org/10.1109/ETFA.2011.6059017.

[30] Corraro, G., Bove, E., Garbarino, L., and Memoli, E., “A novel approach for the development and coding of avionics functionalities for IMA architectures,” *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, 2018, pp. 1–8. https://doi.org/10.1109/DASC.2018.8569824.

[31] Lukić, B., Ahlbrecht, A., Friedrich, S., and Durak, U., “State-of-the-Art Technologies for Integrated Modular Avionics and the Way Ahead,” *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pp. 1–10. https://doi.org/10.1109/DASC58513.2023.10311229.

[32] for Standardization, I. O., Commission, I. E., of Electrical, I., and Engineers, E., *ISO/IEC/IEEE 24765: 2017(E): ISO/IEC/IEEE International Standard - Systems and Software Engineering–Vocabulary*, IEEE Std, IEEE, 2017. URL https://doi.org/10.1109/IEEESTD.2017.8016712.

[33] Joshi, A., Whalen, M. W., and Heimdahl, M. P. E., “Model-Based Safety Analysis Final Report,” Tech. rep., National Aeronautics and Space Administration and Rockwell Collins, 2005. Available at http://shemesh.larc.nasa.gov/fm/papers/Model-BasedSafetyAnalysis.pdf.

[34] Delange, J., Pautet, L., and Kordon, F., “Modeling and Validation of ARINC653 architectures,” *ERTS2 2010, Embedded Real Time Software & Systems*, Toulouse, France, 2010, pp. 1–8. URL https://hal.science/hal-02269428.

[35] Dörr, T., Schade, F., Ahlbrecht, A., Zaeske, W., Masing, L., Durak, U., and Becker, J., “A Behavior Specification and Simulation Methodology for Embedded Real-Time Software,” *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2022, pp. 151–159. https://doi.org/10.1109/DS-RT55542.2022.9932069.

Appendix

```xml
<?xml version="1.0" encoding="US-ASCII"?>
<MODULE xmlns="http://www.aviation-ia.com/aeec/ARINC653/P1S5"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.aviation-ia.co/aeec/ARINC653/
   P1S5 ExampleSchema.xsd" Name="MyModule">
   <Partitions>
      <Partition>
         <PartitionDefinition Name="systemManagement" Identifier="1"
         PartitionHMNameRef="systemManagement HM table" />
         <PartitionPeriodicity Duration="20000000" Period="100000000"/>
         <MemoryRegions>
            <MemoryRegion Type="RAM" Size="1048576" Name="mainMemory"
            AccessRights="READ_WRITE"/>
            <MemoryRegion Type="Flash" Size="524288" Name="Flash"
            AccessRights="READ_ONLY"/>
         </MemoryRegions>
         <PartitionPorts>
            <PartitionPort>
               <QueuingPort MaxMessageSize="30" Name="Stat_2Dq"
               MaxNbMessage="30" Direction="DESTINATION"/>
            </PartitionPort>
            <PartitionPort>
               <QueuingPort MaxMessageSize="30" Name="Stat_3Dq"
               MaxNbMessage="30" Direction="DESTINATION"/>
            </PartitionPort>
         </PartitionPorts>
      </Partition>
   </Partitions>
   <Schedules>
      <PartitionTimeWindow PeriodicProcessingStart="true" Duration="20000000"
      PartitionNameRef="systemManagement" Offset="0"/>
      <PartitionTimeWindow PeriodicProcessingStart="true" Duration="20000000"
      PartitionNameRef="systemManagement" Offset="100000000"/>
   </Schedules>
   <HealthMonitoring>
      <ModuleHM StateIdentifier="1" Description="module init">
         <ErrorAction ErrorIdentifierRef="1"
         ModuleRecoveryAction="SHUTDOWN"/>
         <ErrorAction ErrorIdentifierRef="2"
         ModuleRecoveryAction="SHUTDOWN"/>
         <ErrorAction ErrorIdentifierRef="3"
         ModuleRecoveryAction="SHUTDOWN"/>
         <ErrorAction ErrorIdentifierRef="4"
         ModuleRecoveryAction="IGNORE"/>
         <ErrorAction ErrorIdentifierRef="5"
         ModuleRecoveryAction="IGNORE"/>
      </ModuleHM>
   </HealthMonitoring>
</MODULE>
```

Listing 1 ARINC 653-compliant sample XML configuration

```
function [pars, poar] = transform(m)
   obj = m.Architecture.Components;
   catcher = "";
   stringlit = char("obj.OwnedArchitecture.Components");
   while catcher ~= "Error"
      try get(obj.OwnedArchitecture.Components, "OwnedArchitecture")
         obj = eval(stringlit);
      catch
         catcher = "Error";
      end
   end

   for i = 1:length(obj)
      if getStereotypes(obj(1, i)) == "Integrated_Modular_Avionics.Partition"

         prop = getStereotypeProperties(obj(1, i).Architecture);
         pos = strfind(prop(1), '.');
         pars = [];
         for j = 1:length(prop)
            con = getProperty(obj(1, i).Architecture, prop(j));
            pars = [pars, {[extractAfter(prop(j), pos(end)), con]}];
         end

         opor = obj(1, i).Ports;
         poar = [];
         for j = 1:length(opor)
            prop = getStereotypeProperties(opor(1, j).ArchitecturePort);
            pos = strfind(prop(1), '.');
            port = [];
               for k = 1:length(prop)
                  con = getProperty(opor(1, j).ArchitecturePort, prop(k));
                  port = [port, {[extractAfter(prop(k), pos(end)), con]}];
               end
            poar{j} = port;
         end

         % ...
         % ...
         % Writing to XML file omitted for reasons of conciseness
      end
   end
end
```

Listing 2   MATLAB code for parsing model properties and writing to XML file