Journal of Aerospace Information Systems
Vol. 22, No. 5, May 2025

A Streamlined Approach Toward Automated Generation and Validation of ARINC 653–Compliant Avionics Configurations

Bojan Lukić,* Sven Friedrich,† and Umut Durak‡
German Aerospace Center (DLR), 38108 Braunschweig, Germany

https://doi.org/10.2514/1.I011439

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 only a single function per line-replaceable unit in federated avionics architectures. However, the diversity in IMA solutions limits the portability of applications, which eventually led to the development of IMA standards and guidelines. One such standard is ARINC 653, which outlines, among others, the necessary data for specifying any ARINC 653 configuration. To meet the requirements set by aforementioned standards, this paper proposes a Model-Driven Development approach for engineering ARINC 653–compliant avionics configurations. The authors present a setup for automated configuration, verification, and validation of such configurations. The approach involves the use of MATLAB, System Composer, and Simulink for modeling and configuring the system, as well as generating ARINC 653–compliant configurations in XML format. The paper further exemplifies the approach for modeling and automatically configuring ARINC 653–compliant systems and highlights 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 automate the engineering process and increase the quality of the system.

I. 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 revolutionized 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 federated architectures, in which each system function has 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].

At its heart, IMA incorporates the ARINC 653 standard [2], 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, which prevents interference and ensures 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 [3–6]. This approach leverages 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 and ensure 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 and 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 steps in the overall development process: 1) the use of models as a means for single source of truth development, 2) domain-specific modeling with ARINC 653 profiles, 3) automatic generation of configurations from IMA architecture models, and 4) V&V of these architectures. The remaining paper is structured as follows: In Sec. II, background information on technologies and methodologies is presented. Section III delves into the current state of configuration development for ARINC 653–compliant systems. The main part, Sec. IV, discusses MDD for automatically generating said configurations and possible V&V methods for such models. The results of the work are discussed in Sec. V, before concluding the paper in Sec. VI.

II. Background

IMA contains many functions on the same General Processing Module (GPM) as opposed to a single function per Line-Replaceable Unit (LRU) in federated architectures [1]. As the diversity of the IMA solutions in the avionics industry was moving toward restraining the portability of the applications, several standards were developed, with one of them being ARINC 653.

The recognized acceptable means of compliance for certification of airborne software is the DO-178C/ED-12C [7]. 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 DO-178C/ED-12C, the development of state-of-the-art IMA refers to the deployment of functions of mixed DAL (A–D) inside the same GPM. Further, the standard sets requirements for software partitioning for containing/isolating faults. The specification of software partitioning is standardized in ARINC 653 [2]. ARINC 653–compliant hypervisors are typically used in IMA contexts, with some software items implemented as partitions reaching up to the highest of DAL, i.e., DAL A.

Presented as Paper 2024-3948555 at the 2024 AIAA Science and Technology Forum and Exposition, Orlando, FL, January 8–12, 2024; received 28 February 2024; accepted for publication 10 February 2025; published online Open Access 19 March 2025. Copyright © 2025 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved. All requests for copying and permission to reprint should be submitted to CCC at www.copyright.com; employ the eISSN 2327-3097 to initiate your request. See also AIAA Rights and Permissions www.aiaa.org/randp.

*Research Associate, Institute of Flight Systems; bojan.lukic@dlr.de (Corresponding Author).

†Research Associate, Institute of Flight Systems.

‡Group Leader, Institute of Flight Systems. Associate Fellow AIAA.

The most relevant standards that are of particular interest for this paper are presented in the next sections.

A. DO-297/ED-124

DO-297/ED-124 [1] is a standard for the development and certification of IMA systems. It serves as a primary resource for certification authorities, such as the Federal Aviation Administration (FAA) and European Union Aviation Safety Agency (EASA), in approving IMA systems for flight. The document provides specific guidance and certification considerations for stakeholders involved in the development of IMA platforms, including platform and module suppliers, application suppliers, IMA system integrators, and certification authorities.

One of the core aspects emphasized in DO-297/ED-124 is the adherence to safety requirements. The document provides detailed instructions on how to conduct safety assessments, identify hazards, and mitigate risks associated with IMA systems. It also highlights the importance of thorough testing and validation procedures to ensure the reliability and integrity of the developed systems. By following these guidelines, developers can mitigate potential risks and ensure that IMA systems meet the stringent safety standards required in the aviation industry. Another crucial aspect addressed by DO-297/ED-124 is the use of standardized interfaces, particularly the ARINC 653 standard. In accordance with ARINC 653, DO-297/ED-124 highlights the significance of robust time and space partitioning to ensure the isolation of failures and prevent interference between different applications within the IMA system.

The methods for developing IMA platforms presented in the work on hand are continuously being improved. One of the main considerations during the development of the methods is compliance with different standards applicable to IMA platforms. That is, ETSO-2C153 [8] and ETSO-C214 [9] for compliance of IMA platforms and aircraft parts with minimum performance standards, DO-178C/ED-12C for certification of airborne software, and DO-297/ED-124 for the certification of IMA systems. Lastly, ARINC 653 plays a significant role in IMA platform development regarding the partitioning properties of hypervisors.

B. 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 DO-178C/ED-12C. 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. This includes not only 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 [2]. For the configuration of the static schedule, the most important datum is the *major frame*, which determines the smallest frequency at which partition time windows can be configured. The partition time windows are arranged within the major frame, which repeats periodically, as demonstrated with three partition time windows in Fig. 1.

Just like the major frame, the partitions need to be aligned sequentially to run periodically. This means that only three values are necessary for the configuration of the time window for each partition*: duration*, *offset*, and *period*. The duration specifies the length for each partition time window, 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, there is for partitions 2 and 3. Suppose that the major frame duration in the

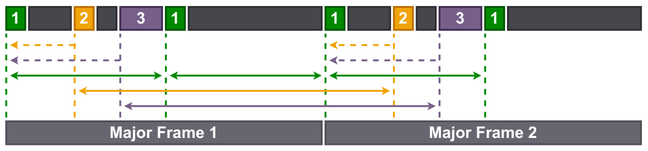

Fig. 1 Static scheduling in ARINC 653. Dashed arrows: offset; normal arrows: period.

setup is 1 s, with the period for partition 1 being 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 interpartition 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 cannot be changed during runtime.

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.

C. ARINC 653–Specific Configuration

According to ARINC 653, configurations are used for general partition specifications, including interpartition 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* [10].

The *Partitions* element describes the partitions configuration in the module and contains the subelement *Partition*. The subelements 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 subelement *PartitionTimeWindow*. The *HealthMonitoring* element represents the health-monitoring configuration for the module. It contains the subelements *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 [2]. 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 that are, according to ARINC 653, explicitly not included in the configuration are the following:

1) Number of associated processor cores—the number of logical processor cores used to schedule processes associated with a partition
2) Operating mode—the partition's execution state
3) 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.

D. 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 isolation within ARINC 653–compliant systems [2].

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" [2]. 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–compliant system [11,12].

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 [13]. 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 [14,15]. For this paper, the XML format is used to provide all necessary information for custom configuration of end systems according to ARINC 653.

E. 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 [16,17].

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 [17]. Metamodeling in MDD involves the following key concepts:

1. *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 [18].

2. *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 [19].

3. *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 [20]. 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 [21,22].

Through metamodeling, models can be created that capture the essential characteristics of a system and its components. Metamodeling also promotes MDD 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 [23].

The Object Management Group (OMG) defines a standard called Meta-Object Facility (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 [24].

The M3 metamodel level is omitted. This paper focuses on the use of custom profiles as metamodels that extend 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:

4. *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 [25–27].

5. *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 special meaning [26,27].

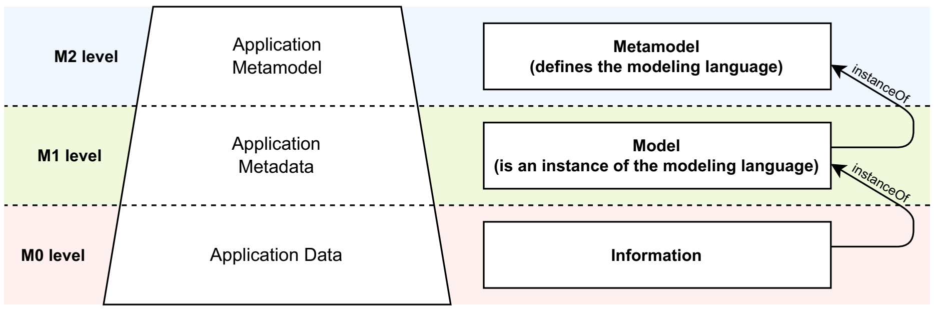

Fig. 2 Metamodeling layers and patterns in MDD.

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 [28,29].

III. Related Work

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

Uludağ et al. [30] 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 reusable 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 be used as a basis for future avionics development projects. With their work, Uludağ et al. tackle some disadvantages from other related works. Their method presents a collaborative model-based systems engineering environment that outperforms solo development practices that do not involve multiple experts.

Lafoz et al. [11] 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 describes 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 an 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. The main advantage of the method proposed in the paper is the coverage of large parts of the ARINC 653 configuration development process. This concerns the automatic generation of code, the definition of tests and required stubs, and the use of qualified tools for verification. The main disadvantage of the method is that it does not extensively address how the proposed methods scale with an increasing number of partitions or applications. The performance overhead associated with context switching and interpartition communication may become more pronounced in larger systems.

Horváth et al. [31] show the application of MDD to the development of ARINC 653 configuration tables in the aeronautic domain in their paper. The paper presents a toolchain that generates configuration tables from high-level architecture models and demonstrates the benefits of using MDD in this context. The authors point 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 work from Horváth et al. excels by addressing current certification challenges for generating ARINC 653 configuration tables. One disadvantage of the paper is that it is mainly tailored for the Wind River VxWorks 653 platform.

The paper [12] by Duprat et al. is about the model-based approach in configuration data management for LVCUGEN Flight Software. The paper shows how this approach is 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 some checker tools and generators. The relevant outcomes in this paper are the model-based approach as an effective way to manage configuration data in generic software frameworks and the assurance of 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. One disadvantage of the paper is that the reliance on formal methods and Object Constraint Language (OCL) requires a certain level of expertise that may not be readily available.

Other publications in this context are [32], which discusses a verification method of end-to-end real-time properties on IMA systems; [33], about the use of the Simulink environment to implement IMA partition models through an ARINC 653 blockset; and [14], 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 systems validation both on the system design and on the software implementation side. Some of the previously mentioned challenges from related work that the work at hand tries to tackle are the generalization of the system development to satisfy different platforms and the scalability of the method using MDD. By using a pragmatic model-to-text transformation, the model-based toolchain is generic enough to conform to different platforms and configuration formats. The different levels of abstraction used for modeling help with scaling the system up, depending on the use case.

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

The method for developing ARINC 653–compliant avionics architectures, presented in this paper, involves two parts: first, modeling the logical architecture of the ARINC 653 system with its components and attributes; second, modeling the functional architecture of the applications. For the first part, the avionics architecture is modeled in System Composer, while for the second part, the functional architecture is modeled in Simulink. For the functional part in Simulink, a complete ARINC 653–compliant system metamodel 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 development of functional system elements, more software tools are available, as shown in a later part of the paper.

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. In a final step, the generated functional model is used as a wrapper, i.e., a template, which can be extended with code and system functions. The workflow is illustrated in Fig. 3.

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

Ultimately, the aim is the automated and seamless integration between system 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 into a functional Simulink model. A sample configuration taken from the ARINC 653 standard is shown in Listing 1.

One of the major advantages of the presented approach is its MDD nature. MDD allows developers to work at a higher level of abstraction, which simplifies the design process. By using models instead of code, teams can focus on system behavior and architecture without getting distracted with implementation details. Using a model-driven approach promotes consistency across the system. Standardized models help ensure that different parts of the system adhere to the same

[§]Arinc 653 blockset, as presented in Ref. [33].

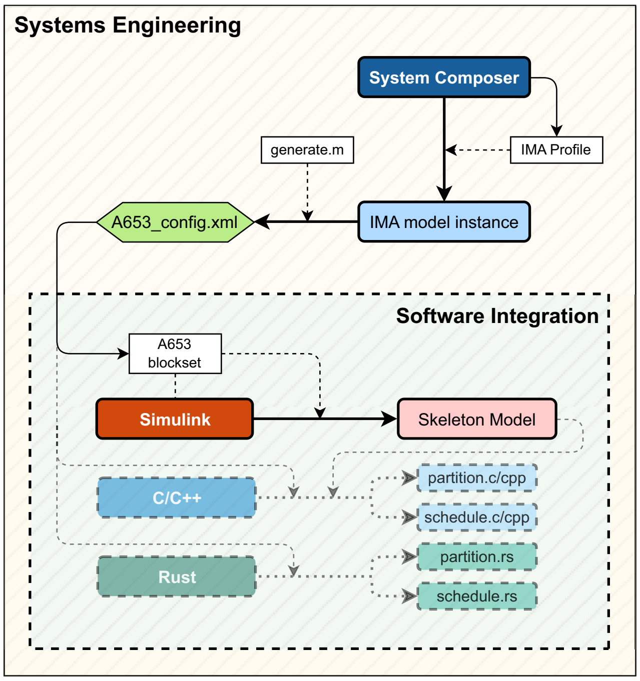

Fig. 3 Workflow followed in this paper portrayed in the systems and software engineering landscape.

design principles and frameworks. This is particularly beneficial when adhering to domain-specific standards, such as the ones mentioned in this paper, that is, DO-297 and ARINC 653.

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 Fig. 4.

GPMs refers to computing modules that execute application software. Essentially, 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 Fig. 5.

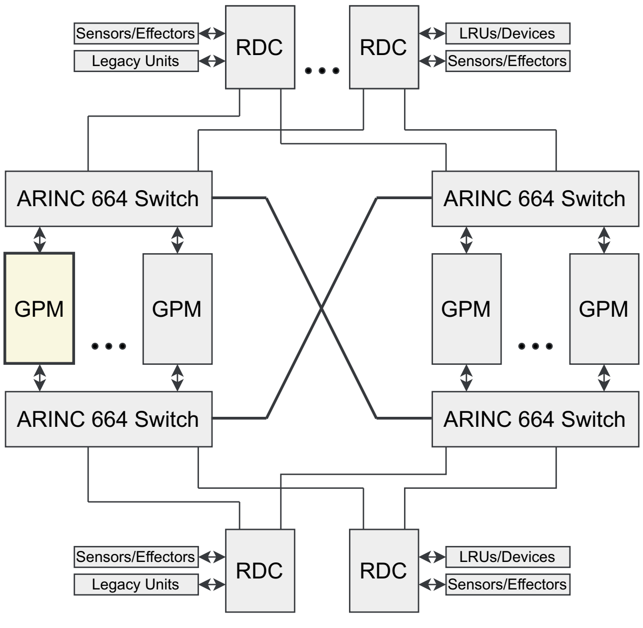

Fig. 4 Integrated system architecture.

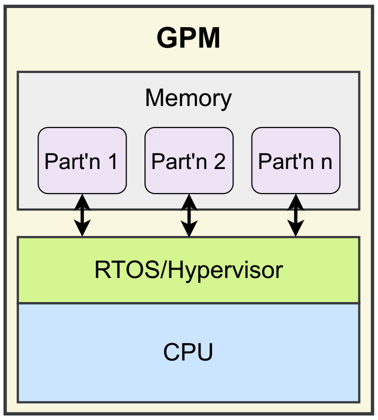

Fig. 5 One GPM with three partitions on discrete memory spaces.

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 set of applications. By means of virtualization, partitions are 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 in the memory is assigned to each partition, assuring spatial separation between partitions [34]. Given that the scope of this paper is to specifically target the configuration of partitions and their schedules, only the GPM inside of an overall IMA architecture is discussed.

The avionics architecture modeled in the 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 managed by the hypervisor. The respective model can be seen in Fig. 6.

The use case for this demonstration covers five different applications, with one dedicated partition for each application. The example is taken from the ARINC 653 standard and demonstrates applications typically found in avionics systems. These applications are *IO Processing*, *IVHM*, *Flight Controls*, *Flight Management*, and *System Management*. As this paper demonstrates the MDD of configurations for running such applications in hypervisors, the exact functionality of the applications is out of scope. Hence, they can be regarded as dummy applications. For a more detailed use case, refer to [35].

By using the System Composer model as a single source of truth artifact, compliance with DO-297 and ARINC 653 can be guaranteed to some degree. For instance, by restricting the model to elements limited to IMA, major development specifications from DO-297 are obeyed. The V&V steps described in the standard are then met with simulation in Simulink. By adding constraints to the model, some fundamental requirements from ARINC 653 can be met. Those are spatial isolation, temporal isolation, and fault coverage. The requirement of spatial isolation is met by ensuring that each modeled partition is contained within some unique memory region. Temporal isolation is ensured with correct scheduling and sequential execution of partitions. Using model correctness, the requirement of fault coverage can be met. Specifics regarding model correctness and verification are discussed in Secs. IV.D and IV.E.

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 most interest. Each stereotype contains specific properties. The properties of *Partition* are listed in Table 1.

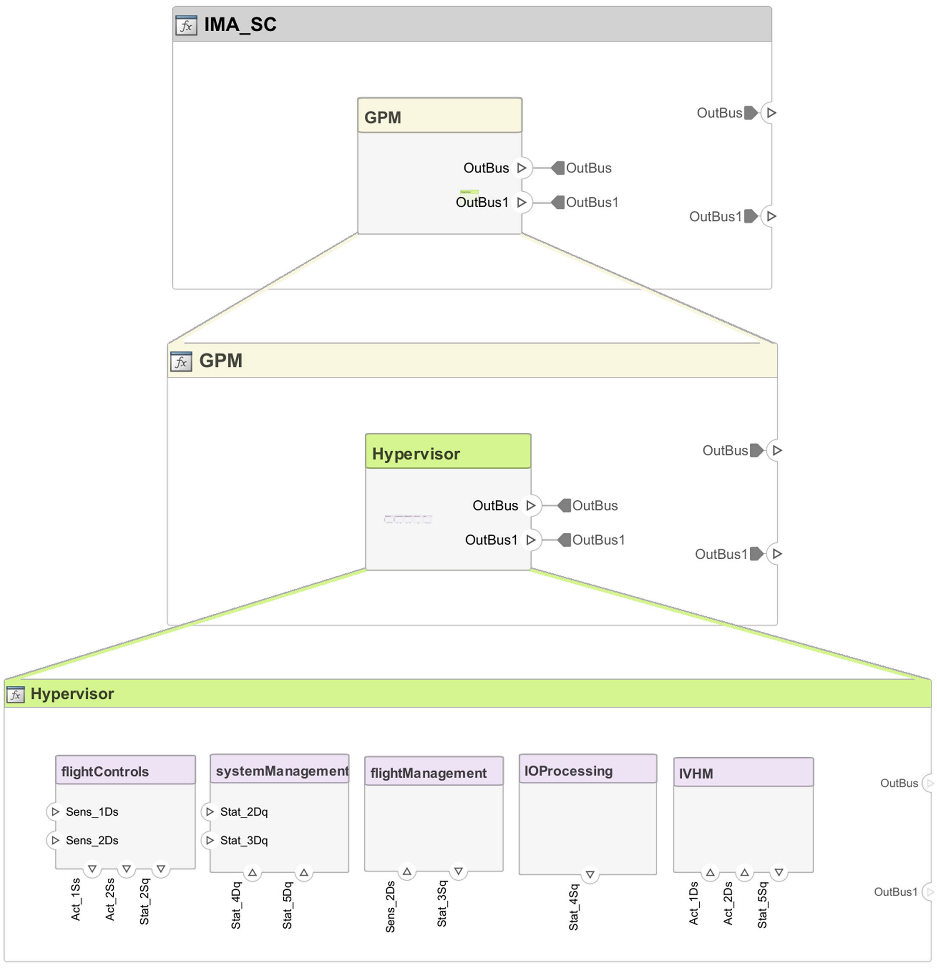

Fig. 6 Three levels of abstraction in System Composer model.

Table 1 Properties contained in the *Partition* stereotype

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

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*.

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

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

The properties included in the 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 and a MATLAB script. 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 Algorithm 1.

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. By applying the script to the System Composer model, the modeled configuration is exported to an XML file. Part of the generated XML file for the use case presented in this paper is illustrated in Listing 1.

C. ARINC 653 Blockset

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

Algorithm 1: Pseudocode for parsing and writing model properties from System Composer to an XML configuration file

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*

execution of multiple applications on a single hardware platform. The ARINC 653 blockset 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 blockset 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 PikeOS[¶] and VxWorks 653[**] code ready to be deployed on the IMA system. The use of a single code generation environment for the automatic code generation of whole partitions (partition infrastructure and applications) satisfies the need to follow a single qualifying process in order to develop safety-critical systems [36].

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, interpartition communication—11 services, intrapartition 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 the Simulink environment. More specifically, the XML configuration file generated from the System Composer model is used as an artifact in the blockset, as shown in Fig. 3. The developed toolbox allows modeling an IMA partition in terms of APEX blocks for partition and process management, intra- and interpartition communication, time management, and health monitoring. The Simulink blocks of the ARINC 653 package are implemented as fully in-lined S-Functions based on the 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 Tool application. Such outputs are integrated into the Simulink infrastructure partition model implemented through the ARINC 653 blockset.

The blockset provides an automated way of generating models from the XML reference file. The functional blocks are then created 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 configurations. For this reason, the continued development of those avionics architectures from configurations 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 activities, such as simulation and 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 [37].

Simulation, an activity in the aforementioned verification process, involves 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 and executing test cases 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 [38].

Taking Fig. 3 as a reference, V&V is performed in the design phase, while simulation and 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. [39] 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 within its designated time window without overlap with other partition time windows; The major frame is consistent and aligned with the defined partition time windows
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. [39] use a custom Architecture Analysis & Design Language (AADL). For the paper at hand, more methods, such as Object Constrain Language (OCL) (see [40]), were identified as potential candidates for verifying model correctness. Moreover, simulation and testing have been conducted within the software integration part of the discussed setup. With simulation, for instance, the runtime analysis on the end systems can be verified.

[¶]PikeOS, as presented in: SYSGO, "PikeOS Certifiable RTOS & Hypervisor," 2024, https://www.sysgo.com/pikeos.

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

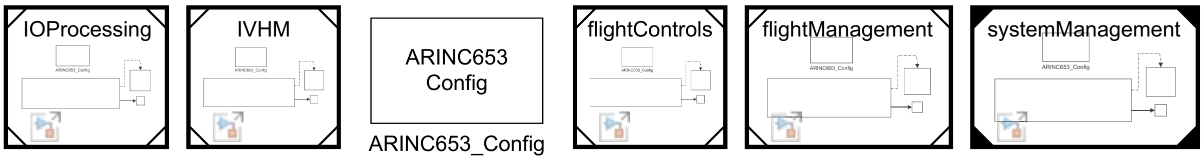

Fig. 7 ARINC 653 blockset model from generated XML configuration.

E. Simulation and Testing

The Simulink environment of the ARINC 653 blockset offers a variety of simulation properties. The developed ARINC 653 system is imported through a generated XML configuration file, like the one exemplified in Listing 1. Each partition is treated as a subsystem with distinct features in the Simulink environment. Each partition subsystem 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. Figure 7 shows the ARINC 653 blockset model created from the configuration that was modeled in Fig. 6.

The partition itself, with its designated execution duration, periodicity, priority, and deadline behavior, is treated as a process. Those parameters are handled within a separate block inside each subsystem. Another level of abstraction deeper lies the functional environment with the specified ports and interfaces. The code that can be generated from the model is compliant with the ARINC 653 API and is treated as a skeleton model that can be extended. Custom applications are either added directly in the functional environment of each partition or in the code skeleton itself.

An important characteristic of ARINC 653 is the time isolation of partitions. Each partition executes within its designated partition time window and only resumes execution within its following window. In the blockset model, a counter is added to each partition to track executions over time. The signals from the counter are logged and plotted after a simulation run. This is to verify that each partition is executing only within its designated schedule and only for as many times as is intended within a certain time frame. Figure 8 shows an exemplary plot with the number of executions over time for each partition in the developed system.

The partitions have different execution frequencies: *IOProcessing*—200 executions, *IVHM*—100 executions, *flightControls*—66 executions, *flightManagement*—50 executions, and *systemManagement*—33 executions, over a time period of 10 s. The frequency of executions is dependent on the *PartitionPeriodicity Period* parameter specified in the developed configuration for each partition.

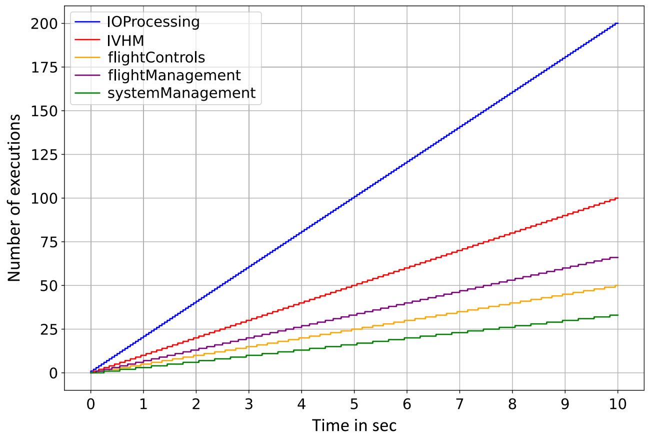

Fig. 8 Plot showing the number of executions over time for the five partitions *IOProcessing*, *IVHM*, *flightControls*, *flightManagement*, and *systemManagement*.

Other properties to be verified through simulation are, for instance, the computational speed and the adherence to deadlines for partitions running custom applications. The computational speed refers to the speed at which partitions compute custom applications assigned to them. That is, for example, how many major frames a partition requires to finish computing single lines of code or functions from assigned applications, or even the whole application. Simulating these properties presumes the same computational speed in the simulation software as in the hardware on the end system. Adherence to deadlines refers to the temporal boundaries of partitions when running processes. According to ARINC 653, a partition cannot overrun its specified partition duration. The running process has to be paused and can only continue at the start of its subsequent partition time window. A typical deadline exception occurring with faulty partitions tested on hardware is, for instance, `P4_TRAP_DEADLINE`. The simulation of deadline exceptions requires the simulation software to have deadline exception handler functionality consistent with ARINC 653.

It is important to note that the aforementioned simulations are dependent on hardware and software specifications and characteristics and might not reflect the same behavior as observed when testing the system on the final end system. For this work, the timing simulation of partitions is out of scope but will be considered for future work.

V. Discussion

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

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. Moreover, MDD allows for improved development efficiency by offering a higher level of abstraction and consistency across the system by developing with standardized models. 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 the presented methods can reduce the development and verification time of IMA platforms.

Major advantages of the work proposed here are twofold. The pragmatic script used for model-to-text transformation enables the system developer to extract information in a concise format. With some extensions, the transformation can be applied easily for different platforms and formats. Also, the method is scalable: on the one hand, for modeling multiple levels of abstractions within the system; on the other hand, for an increasing number of system parameters within the single architecture levels. Compared to related studies, the work on hand excels by being generic enough to be used for different platforms but specific enough within its domain to meet fundamental requirements set by ARINC 653 and related standards.

Moreover, the method is pragmatic and does not rely on major expertise by the user.

VI. Conclusions

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 an 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. The generality of the proposed method enables the extension for other formats and system parts without major adjustments to the underlying processes. The scalability allows developers to extend the modeled IMA system while keeping noticeable efficacy and efficiency for generating configurations.

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 avionics systems. The use of generic profiles and automated configuration generation can further facilitate the creation of various architecture instances from a single source of truth. Some of the works that will be researched further in the future are the real-time simulation capabilities of the ARINC 653 blockset as well as verification processes for the development of avionics systems set at design time.

Appendix: Referenced Configuration Instance and Model Parsing Script

```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_new.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.

References

[1] RTCA and EUROCAE, “Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations,” *Standard, Radio Technical Commission for Aeronautics and European Organization for Civil Aviation Equipment*, Washington, D.C., Nov. 2005.

[2] ARINC, *Avionics Application Software Standard Interface Part 1 Required Services*, Standard, Aeronautical Radio Inc., Bowie, MD, Dec. 2019.

[3] Annighöfer, B., “Model-Driven Development and Simulation of Integrated Modular Avionics (IMA) Architectures,” *SNE Simulation Notes Europe*, Vol. 28, No. 2, 2018, pp. 61–66. https://doi.org/10.11128/sne.28.tn.10414

[4] Annighöefer, 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)*, Inst. of Electrical and Electronics Engineers, New York, 2020, pp. 1–10. https://doi.org/10.1109/DASC50938.2020.9256726

[5] Halle, M., and Thielecke, F., “Next Generation IMA Configuration Engineering - From Architecture to Application,” *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, Inst. of Electrical and Electronics Engineers, New York, 2015, pp. 6B2-1–6B2-13. https://doi.org/10.1109/DASC.2015.7311445

[6] 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*, Shaker, Aachen, 2019, pp. 401–410.

[7] 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.

[8] EASA, “INTEGRATED MODULAR AVIONICS (IMA) PLATFORM AND MODULES,” Standard, European Union Aviation Safety Agency, Cologne, Germany, May 2016.

[9] EASA, “FUNCTIONAL ETSO EQUIPMENT USING AN ETSO-2C153-AUTHORISED IMA PLATFORM OR MODULE,” Standard, European Union Aviation Safety Agency, Cologne, Germany, Aug. 2018.

[10] Lukić, B., Beck, J., and Durak, U., “Unifying Avionics System Configuration: A Concept for Capturing Essential Elements From Computing Resources to Network Topology in a Universal Format,” *AIAA SCITECH 2025 Forum*, AIAA Paper 2025-2515, 2025. https://doi.org/10.2514/6.2025-2515

[11] 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)*, Hyper Articles en Ligne (HAL), Lyon, 2008. https://insu.hal.science/insu-02270111.

[12] 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*, European Space Agency (ESA), Paris, 2021, https://indico.esa.int/event/386/contributions/6235/.

[13] Wilson, A., and Preyssler, T., “Incremental Certification and Integrated Modular Avionics,” *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, Inst. of Electrical and Electronics Engineers, New York, 2008, pp. 10–15. https://doi.org/10.1109/DASC.2008.4702768

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

[15] Eu-Teum, C., Ha, O.-K., and Jun, Y.-K., "Configuration Tool for ARINC 653 Operating Systems," *International Conference of Multimedia and Ubiquitous Engineering*, Vol. 9, No. 4, 2014, pp. 73–84. https://doi.org/10.14257/ijmue.2014.9.4.08

[16] 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

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

[18] Favre, J.-M., and NGuyen, T., "Towards a Megamodel to Model Software Evolution Through Transformations," *Electronic Notes in Theoretical Computer Science*, Vol. 127, No. 3, 2005, pp. 59–74. https://doi.org/10.1016/j.entcs.2004.08.034

[19] Bézivin, J., and Gerbé, O., "Towards a Precise Definition of the OMG/MDA Framework," *Automated Software Engineering*, Inst. of Electrical and Electronics Engineers, New York, 2001, pp. 273–280. https://doi.org/10.1109/ASE.2001.989813

[20] 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.

[21] Neto, V. V. G., "A Simulation-Driven Model-Based Approach for Designing Software Intensive Systems-of-Systems Architectures," Ph.D. Thesis, Université de Bretagne Sud; Universidade de São Paulo (Brésil), São Paulo, June 2018, https://theses.hal.science/tel-02146340v1/document.

[22] Mens, T., and Van Gorp, P., "A Taxonomy of Model Transformation," *Electronic Notes in Theoretical Computer Science*, Vol. 152, March 2006, pp. 125–142. https://doi.org/10.1016/j.entcs.2005.10.021

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

[24] 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

[25] Fuentes, L., and Vallecillo, A., "An Introduction to UML Profiles," *UPGRADE, The European Journal for the Informatics Professional*, Vol. 5, No. 2, April 2004, pp. 9–11.

[26] Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed., Addison-Wesley, Boston, MA, 2003, p. 90.

[27] Selic, B., "UML 2.0 Superstructure Specification," TR ptc/04-10-02, Object Management Group (OMG), Milford, 2005, http://www.omg.org/cgi-bin/doc?ptc/2004-10-02.

[28] Giachetti, G., Marín, B., and Pastor, Ó., "Integration of Domain-Specific Modelling Languages and UML Through UML Profile Extension Mechanism," *International Journal of Computer Applications*, Vol. 6, 2009, pp. 145–174, https://api.semanticscholar.org/CorpusID:7952602.

[29] Abouzahra, A., Bézivin, J., Fabro, M. D. D., and Jouault, F., "A Practical Approach to Bridging Domain Specific Languages with UML Profiles," SoftMetaWare, Oneroa, 2005, https://api.semanticscholar.org/CorpusID:5620908.

[30] 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)*, Inst. of Electrical and Electronics Engineers, New York, 2023, pp. 1–11. https://doi.org/10.1109/DASC58513.2023.10311115

[31] Horváth, A., and Varró, D., "Model-Driven Development of ARINC 653 Configuration Tables," *29th Digital Avionics Systems Conference*, Inst. of Electrical and Electronics Engineers, New York, 2010, pp. 6.E.3-1–6.E.3-15. https://doi.org/10.1109/DASC.2010.5655451

[32] Lauer, M., Ermont, J., Boniol, F., and Pagetti, C., "Latency and Freshness Analysis on IMA Systems," *ETFA2011*, Inst. of Electrical and Electronics Engineers, New York, 2011, pp. 1–8. https://doi.org/10.1109/ETFA.2011.6059017

[33] 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)*, Inst. of Electrical and Electronics Engineers, New York, 2018, pp. 1–8. https://doi.org/10.1109/DASC.2018.8569824

[34] 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)*, Inst. of Electrical and Electronics Engineers, New York, 2023, pp. 1–10. https://doi.org/10.1109/DASC58513.2023.10311229

[35] Lukić, B., Nöldeke, P., Durak, U., Klimmek, M., Jakob, S., and Gläser, M., "From Architecture Models to a Target Platform: A Systematic Approach to Model-Driven Development of Dynamically Reconfigurable Avionics Systems," *2024 IEEE/AIAA 43rd Digital Avionics Systems Conference (DASC)*, Inst. of Electrical and Electronics Engineers, New York, 2024, pp. 1–10. https://doi.org/10.1109/DASC62030.2024.10749454

[36] Lukić, B., Friedrich, S., Schubert, T., and Durak, U., "Automated Configuration of ARINC 653-Compliant Avionics Architectures," *AIAA SCITECH 2024 Forum*, AIAA Paper 2024-1856, 2024. https://doi.org/10.2514/6.2024-1856

[37] International Organization for Standardization and International Electrotechnical Commission, and Institute of Electrical and Electronics Engineers, *ISO/IEC/IEEE 24765: 2017(E): ISO/IEC/IEEE International Standard - Systems and Software Engineering–Vocabulary*, Inst. of Electrical and Electronics Engineers, New York, 2017. https://doi.org/10.1109/IEEESTD.2017.8016712

[38] Joshi, A., Whalen, M. W., and Heimdahl, M. P. E., "Model-Based Safety Analysis Final Report," National Aeronautics and Space Administration and Rockwell Collins TR DOT/FAA/TC-20/42, National Aeronautics and Space Administration (NASA), Washington, D.C., 2005. http://shemesh.larc.nasa.gov/fm/papers/Model-BasedSafetyAnalysis.pdf.

[39] Delange, J., Pautet, L., and Kordon, F., "Modeling and Validation of ARINC653 Architectures," *ERTS2 2010, Embedded Real Time Software & Systems*, Hyper Articles en Ligne (HAL), Lyon, 2010, pp. 1–8, https://hal.science/hal-02269428.

[40] 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)*, Inst. of Electrical and Electronics Engineers, New York, 2022, pp. 151–159. https://doi.org/10.1109/DS-RT55542.2022.9932069

J. Jeannin
Associate Editor