Unifying Avionics System Configuration: A Concept for Capturing Essential Elements from Computing Resources to Network Topology in a Universal Format

Bojan Lukić[*] Janick Beck[†] Umut Durak[‡]
German Aerospace Center (DLR), Braunschweig, Germany

Configurations enable a system integrator to set system parameters according to functionality requirements. The same is true for avionics software which can be configured for specific needs like application execution frequency and execution duration. Configurations come in different forms. Most software systems use standardized formats, such as the Extensible Markup Language (XML), to deposit essential configuration parameters according to which the system shall operate. The current gap in systems configuration for avionics is twofold. First, there is a lack of a uniform configuration format. Second, essential system resources are rarely captured in a centralized and standardized format. This paper addresses these two challenges. Here, the authors present the concept of a unified avionics system configuration. The system configuration concerns the computing resources as well as the network backbone typically found in Integrated Modular Avionics (IMA) systems. The research presented here explores the essential elements of computing modules and network protocols of IMA systems. Then, a suitable serialization format is selected to capture a superset of these elements. The results show that the approach of unifying avionics configurations can enhance the development of these systems by making the engineering process more flexible and coherent.

I. Nomenclature

IMA	=	Integrated Modular Avionics
GPM	=	General Processing Module
CPU	=	Central Processing Unit
XNG	=	Xtratum Next Generation
AFDX	=	Avionics Full-Duplex Switched Ethernet
HSDN	=	High-Speed Data Network
XML	=	Extensible Markup Language
JSON	=	JavaScript Object Notation
YAML	=	YAML Ain't Markup Language
TOML	=	Tom's Obvious Minimal Language

II. Introduction

With continuous improvements in computing paradigms, avionics systems enjoy a steady extension of capabilities. One example is the current ARINC 653 development practice [1] which provides modularity and safety for avionics applications. We expect that the next generation of avionics will contain highly distributed modules with dynamic resource allocation overarching multiple modules within one system. While these developments introduce new possibilities for general aircraft functionalities and designs, they also come with an increase in system complexity. Previously separate avionics modules now become interconnected and complex dependencies occur. A concrete example is the serverless avionics concept. In this architecture, applications are not bound to one General Processing Module (GPM) anymore, but flexibly assigned to available computing resources within the whole avionics system by the platform. At the same time, a real-time network backbone must ensure that information can still be routed to modules with dependencies without violating deadlines.

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

Simultaneously, the configuration of the aforementioned systems is also increasing in complexity. Apart from common parameters, the complete composition of the system needs to contain at least some dependencies found between modules within the system. Because the mapping of system components is usually defined in early steps of the development process, design decisions need to be chosen with care and configurations need to be coherent with respect to the intended system behavior. What makes the configuration development process unnecessarily challenging for some avionics systems is the repetition of the same information in various configuration artifacts, distinction of the configuration artifacts for various aspects, and inconsistency of configuration formats. While for some systems, configurations are developed according to prevalent development examples, most integrators deploy their own specification patterns and configuration parameters. Also, in some cases, configurations are not stored centrally but rather split between different files which potentially decreases their tracing proficiency and consistency.

This paper addresses these challenges by exploring two notions. The first concerns the common properties found in avionics systems using different computing modules and network protocols. By establishing a superset of essential avionics attributes, a universal configuration format can be defined. To achieve this universal format, different hypervisors and network technologies are explored and compared with the goal of unifying their properties. The second notion concerns the serialization format that represents the final configuration instance of an avionics system. Here, different serialization formats are compared to identify their suitability for capturing a unified avionics configuration. The remaining paper is structured as follows: Some essential information on the discussed topic is presented in section III. Some supporting literature is discussed in section IV, before coming to the main part of the paper in section V. Results are discussed in section VI and the paper is concluded in section VII.

III. Background

The following paragraphs try to establish a common understanding of underlying technologies and concepts discussed in this paper. The definitions listed here are either taken from [2] or from the referenced avionics standards and guidelines.

The underlying avionics concept assumed during the development of a unified configuration format in this paper is the state-of-the-art Integrated Modular Avionics (IMA). The proposed concepts presented in this paper are flexible and can be applied independent of the used technology. Nevertheless, some common practices from the IMA domain, such as ARINC 653, are reflected in the configuration. Because of the universal nature of the sought-after configuration, the configuration can be edited and extended with new parameters rapidly and in a centralized way. This is useful when considering the integration of additional hypervisors or network protocols, which contain new and unique parameters, in an existing system.

Integrated Modular Avionics

IMA refers to an avionics architecture used in aircraft that consolidates multiple avionics functions into a single, modular system. This system comprises various computing modules that operate within a real-time network, allowing for the support of diverse applications with varying levels of criticality.

Processing Module

A GPM, or short *Processing Module*, is a computational unit that performs data processing and supports various avionics functions, such as flight control, navigation, and system management. It is designed to handle multiple tasks within a single hardware module. The GPM interfaces with other system components, providing a flexible and scalable architecture for modern avionics.

Hypervisor

A hypervisor is a specialized software layer that enables the creation and management of virtual software environments within avionics systems. It allows multiple partitions, i.e., applications, to run concurrently on a single physical hardware platform, facilitating the separation of critical flight control functions from non-critical applications. Hypervisors can be deployed in GPMs found within IMA systems. They are a critical part of avionics configuration.

Avionics Network

Avionics networks discussed in this paper refer to the interconnected communication frameworks that serve as the backbone for data exchange among various avionics modules of an IMA system. These networks are designed to support real-time operations, ensuring that critical flight data and control commands are transmitted with minimal latency and high reliability. One example of such a network is the Avionics Full-Duplex Switched Ethernet (AFDX) network that is defined by, and an implementation of, the ARINC 664 standard [3].

Data Serialization

Data serialization is the process of converting complex data structures into a format that can be easily stored, transmitted, and reconstructed. This transformation allows data to be efficiently saved to physical devices or sent over networks, facilitating communication between different systems or applications. Common serialization formats are JavaScript Object Notation (JSON) or Extensible Markup Language (XML), which are used for instance in web services. Serialization is essential for ensuring that data maintains its integrity and structure during storage and transmission.

A detailed description of the above-mentioned technologies in a cohesive system is described in [4].

IV. Related Work

Literature that compares universal system configurations in different serialization formats is sparse, even more so when narrowing down to the field of avionics. In the field of avionics, there seemingly is an implicit consensus to acknowledge configuration formats commonly used by technology vendors or defined in standards and guidances. Alternatively, custom formats in domain-specific environments are developed and used. For instance in [5], Chrysalidis et al. discuss the AvioNET concept. AvioNET is a seamless tool chain to improve the development, integration, and testing of avionics platforms. A highly automated and interconnected tool chain addresses the complexity of avionics development. The paper introduces the Universal Configuration Format (UCoF) which offers a target-independent configuration of avionics platforms. UCoF offers certain advantages such as the integration on virtual as well as real hardware, and the standardization of data storage. AvioNET, along with its UCoF, has some drawbacks. The reliance on standardized data models means that inconsistencies can arise if industry standards are not applied consistently. In addition, AvioNET is tool-dependent and cannot be deployed easily in generic development environments without its dedicated and tailor-made tools.

The remaining related work presented in the next paragraphs comes from domain-independent research. While the publications are not within the domain of aviation, the presented literature provides general information regarding data serialization formats for configurations.

The research presented in [6] offers a comprehensive comparison of YAML Ain't Markup Language (YAML) and JSON as data serialization formats. According to the authors, YAML's advantages include its support for complex features like object references and data type tagging, which facilitate direct mapping to native programming objects. On the other hand, JSON excels in performance, being faster in both serialization and deserialization, which makes it suitable for applications where speed is critical. Its simpler syntax allows for easier generation and parsing, aligning with its goal as a straightforward data exchange format. Nevertheless, JSON has limitations, such as a lack of support for advanced features like object references and explicit data types, which can restrict its use in more complex scenarios. The authors do not discuss other serialization formats and testing is done domain-independent.

The authors in [7] investigate four data serialization formats – XML, JSON, Thrift, and ProtoBuf. They focus on their performance in mobile environments characterized by limited resources and bandwidth. The authors conducted tests on an Android device, measuring serialization and deserialization speeds as well as data sizes for both text-heavy and number-heavy objects. XML is noted for its human readability and extensive documentation, making it easy to debug and suitable for complex data structures. However, its verbosity results in larger data sizes and slower performance, making it less ideal for mobile applications. JSON, on the other hand, is lightweight and efficient, offering a balance of readability and performance. It is widely supported across platforms, which enhances its usability in mobile development, though it has limitations in extensibility and input validation. In contrast, Thrift and ProtoBuf, as binary formats, excel in speed and efficiency, particularly for number-heavy data. However, their lack of human readability complicates debugging and manual data entry, and they require compilation, which can hinder interoperability with existing web services.

Other relevant work, which is, however, outside the aviation domain, is [8–11]. In the context of the related work presented above, the paper at hand tries to make a scientific contribution by comparing four data serialization formats, namely XML, JSON, YAML, and Tom's Obvious Minimal Language (TOML), and motivating the use of a specific and standardized data format for a universal avionics configuration. Additionally, it demonstrates what configuration parameters are needed to capture this universal and platform-independent avionics configuration.

V. Unifying Avionics System Configuration

In the search for a unified avionics system configuration, different configuration aspects need to be considered. On the one hand, the mechanism of the system itself needs to be captured. On the other hand, an appropriate format for storing the information needs to be explored. This section discusses two integral technologies which are generally found in IMA systems. One is the computing resources managed by hypervisors, the other the real-time avionics networks. Afterwards, it explores possibilities of storing common information for the two technologies in a standardized and serialized format. Towards the end of the section, using a practical example, the authors propose a universal configuration format. This format captures all essential elements, from the computing resources to the network topology, found onboard an avionics system.

A. Hypervisors

This paper exemplifies the proposed configuration methodology with state-of-the-art avionics technologies. An IMA architecture is used for exemplification to make the proposed methods more tangible. One of the major standards explaining the mechanisms of computing resources deployed in IMA architectures is ARINC 653. For this reason, mostly hypervisors with at least some level of ARINC 653 support were selected. Also, the performance, maturity, and utilization level were factored during selection. The next paragraphs present the selection of hypervisors along with their essential configuration parameters.

Selection of Generic Avionics Hypervisors

DDC-I Deos excels in deterministic real-time performance, making it ideal for safety-critical applications in aerospace and automotive sectors. It supports a wide range of development standards, including DO-178C and ISO 26262. Deos emphasizes timing guarantees and resource management in safety-critical environments, focusing on embedded applications that require strict adherence to performance metrics [12].

SYSGO's PikeOS features a microkernel architecture that enables high modularity and separation of applications, allowing mixed-criticality systems to run simultaneously on a shared computing platform. Its design enhances security and reliability while supporting multiple operating systems. Unlike Deos, PikeOS focuses on providing a robust separation kernel that isolates applications to prevent interference, making it suitable for diverse industrial sectors beyond just aerospace and automotive [2, 13, 14].

Xtratum Next Generation (XNG) is an open-source hypervisor that prioritizes flexibility and security for real-time applications. Its partitioning approach ensures that different operating systems can run concurrently on the same hardware, promoting resource efficiency. XNG's open-source nature distinguishes it from Deos and PikeOS, enabling greater customization and community-driven development, while still catering to safety-critical applications in aerospace and other fields [15, 16].

Some fundamental characteristics of the three hypervisors, with an emphasis on configurability, are shown in Table 1.

	Deos	**PikeOS**	**XNG**
Configuration format	XML	XML	XML
Primitives / Root Elements	logicalMemoryPools, usedFeatureSets, processInstances, threadTemplates, threadNames	Partition, Schedules, HealthMonitoring,	major_frame, partitions, channel, QueuingChannel, SamplingChannel, Schedule, Module
No. of config. files	Single file	Single file	Multiple files
Standard language	C++	C, C++	C
Compilation support	C, C++, Ada95	C, C++	C, C++, Ada95
ARINC 653 support	Supported	Compliant	Compatible

Table 1 Properties of the selected hypervisors *Deos*, *PikeOS*, and *XNG*.

ARINC 653 proposes the three primitives *Partition*, *Schedules*, and *HealthMonitoring* for the configuration of hypervisors. Each primitive type generally contains additional sub parameters that specify technicalities of the hypervisor. Depending on the technology and implementation, above primitives can be expanded so the configuration of a hypervisor can be more extensive than described in ARINC 653. From the three hypervisors described in this paper, only PikeOS follows the exemplary configuration format described in ARINC 653. Deos and XNG both have either a more extensive set of attributes or number of files for configuration. Table 2 gives an overview of the primitives and underlying parameters of the three hypervisors Deos, PikeOS, and XNG. Some branches contain the unique names of the parameters specified in the configuration files of each hypervisor. Capitalization and plural forms are therefore conditioned by these unique names.

- `processTemplate`
  - `logicalMemoryPools`
    - **`pool`** – memory region size
  - `usedFeatureSets`
    - **`usedFeatureSet`**
      - **`usedFeature`** – used process/application
  - `processInstances`
    - **`processInstance`**
      - **`ownedMemoryMappedResources`** – memory for process
      - **`autoCreatedMemoryObjects`** – initialization of process/application; assignment of access rights
  - `threadTemplates`
    - **`threadTemplate`** – schedule with budget
  - `threadNames`
    - **`threadName`** – Central Processing Unit (CPU) cores

- `Module`
  - `Partition`
    - **`PartitionDefinition`** – name and ID of the partition
    - **`PartitionPeriodicity`** – partition periodicity
    - **`MemoryRegions`** – memory region mapped to the partition
    - **`PartitionPorts`** – ports of the partition
  - `Schedules`
    - **`PartitionTimeWindow`** – duration and offset of a partition to be executed
  - `HealthMonitoring`
    - **`SystemErrors`** – list of system errors
    - **`ModuleHM`** – module level HM table
    - **`MultiPartitionHM`** – partition level HM table for multiple partitions
    - **`PartitionHM`** – partition HM table

module.xml

- `Module`
  - `Hypervisor`
    - `id`
  - `Partitions`
    - `MultiPartition-HMTables`
    - `Schedules`
    - `Channels`

channels.xml

- `Channels`
  - `QueueingChannel`
    - `Source`
    - `Destination`
  - `SamplingChannel`
    - `Source`
    - `Destination`

schedules.xml

- `Schedules`
  - `Schedule`
    - `cpu`
      - `Partition-Window`

Table 2 Primitives and parameters for the configurations of the three hypervisors *Deos* (top left), *PikeOS* (top right), and *XNG* (bottom).

As mentioned above, PikeOS follows the exemplary configuration format described in ARINC 653. In Deos, the primitives *processInstances* and *threadTemplates* are equivalent to *Partition* and *Schedules*, respectively. Additionally, Deos specifies the processes/applications related to a partition and the number of CPU cores used in the processing module. XNG splits its configuration into multiple files with the *Module* element being specified in the *module.xml*, *Channels* in the *channels.xml*, and *Schedules* in the *schedules.xml* file. This demonstrates the current challenge in avionics configuration. Even though different technologies are compliant with respective standards, like in this case all hypervisors support ARINC 653 at least to some extent, the configuration strategies are not always consistent. This also applies to other system aspects like the network configuration, which is demonstrated in a later subsection. In the next paragraph, common hypervisor configuration elements are presented and selected for a uniform avionics configuration.

Common Hypervisor Configuration Elements

To understand the commonalities between the inspected hypervisors, the overall configuration parameters for each hypervisor are mapped and compared against one another. This allows us to identify which common configuration parameters currently exist, which parameters are missing in some hypervisors, and what a universal avionics configuration can look like. Table 3 shows a comparison of the three selected hypervisors with their configuration parameters. Similarly to Table 2, the capitalization and plural forms in the following table are conditioned by the unique names of the parameters taken from the configurations.

	Deos	**PikeOS**	**XNG**
Schedules	🟨 *Self-organized*	🟩 Schedules	🟩 Schedule
Error handling	🟨 UndefinedAction	🟩 HealthMonitoring	🟩 HmTable
Partition periodicity	🟩 period	🟩 PartitionPeriodicity	🟩 period
Partition duration	🟩 budget	🟩 Duration	🟩 duration
Memory regions	🟩 ownedMemory	🟩 MemoryRegion	🟩 MemoryAreas
Ports	🟨 *Unspecified*	🟩 PartitionPort	🟩 channel/Channels
Application reference	🟩 processInstance	🟥 *N/A*	🟩 Partition hRef
CPU parameters	🟨 scheduleName	🟥 *N/A*	🟩 Cpu
Reconfiguration	🟩 *Supported*	🟩 *Design time*	🟩 *HM event based*

Table 3 Comparison of the configuration parameters and properties of the selected hypervisors.
🟩 Fully supported 🟨 Undetermined 🟥 Not supported

In this use case, XNG covers all functionalities found in both the PikeOS and Deos hypervisors. Therefore, the superset of configuration parameters selected for a generic hypervisor configuration supports at least the parameters found in XNG configurations. The common configuration format for hypervisors unified with the network configuration is shown in a later part of the paper, in paragraph V.D.

Similarly to this subsection, the next subsection inspects generic avionics network protocols and their common configuration parameters in a similar approach.

B. Networks

Avionics networks must meet several critical properties to ensure reliable and efficient operation in aviation environments. First, they should provide robust fault tolerance to maintain functionality despite potential failures, thereby enhancing safety and operational continuity. Additionally, low latency and high bandwidth are essential for real-time data transmission, which is vital for avionics systems that depend on timely information for navigation and control. Scalability enables the network to accommodate an increasing number of devices and applications as technology evolves [17, 18]. For this paper, the two network protocols AFDX, and High-Speed Data Network (HSDN) are used for demonstration as they meet desired network protocol requirements. Those are networks in the context of aerospace applications that are real-time capable, have a high bandwidth, and are deterministic. It is important to note that when talking about networks, this paper assumes networks which are commonly used as network backbones in IMA architectures. Such an architecture is demonstrated in more detail in [4].

Selection of Generic Avionics Network Protocols

AFDX, adopted from ARINC 664, is a deterministic network protocol widely used in modern avionics systems. It is designed to support the transmission of safety-critical data over Ethernet-based networks while ensuring time-sensitive communication and high reliability. AFDX operates by segmenting data into fixed-size frames and utilizing virtual channels to provide guaranteed bandwidth and latency, which are essential for mission-critical applications such as flight control and navigation systems. Its ability to handle multiple data streams efficiently makes it a foundational technology for the integration of various avionics systems in commercial and military aircraft [3, 4].

An alternative avionics network protocol is HSDN. HSDN is an advanced communication protocol used in aviation to facilitate high-throughput data exchange between onboard systems. HSDN enhances the capabilities of traditional avionics data networks by offering increased bandwidth and reduced latency, enabling real-time data sharing and improved situational awareness. This protocol supports a range of applications, including video streaming, telemetry, and maintenance data transfer, thereby contributing to the overall efficiency and safety of modern aviation operations. By leveraging new technologies, HSDN plays a critical role in the evolution of aviation communication systems [19, 20].

Common Network Configuration Elements

Comparably to the overview for hypervisors in Table 2, Table 4 shows the primitives and parameters contained in configurations for the two network protocols.

- `Global`
  - `TDMA cycle time`
  - `Target hop latency`
- `Network Access Points (NAPs)`
  - `ID`
  - `Time master priority`
  - `Maximum number of hops`
  - `Error reporting`
  - `Authentication`
  - `Extension layers`
  - `Domain Interfaces`
    - `Number of inner domain interfaces`
    - `Number of outer domain interfaces`
    - `Outer domain interfaces`
  - `Outer Domain to Virtual Link (VL) Map`
    - `VL-ID`
    - `Replicability, retransmittability`
  - `VL to Outer Domain Map`
    - `VL-ID`
    - `Deduplication method`
    - `Data flow control`
  - `Inner Domain VL Routing`
    - `VL-ID`
    - `Routes`
  - `NAP TDMA Schedule`
    - `Interface ID`
    - `Entries (VL-ID, start time, traffic type, duration)`

- `Positions`
  - `End System`
    - **`Ports`**
      - L_{max}
      - L_{min}
  - `Filtering, Policing, and Forwarding`
    - **`Virtual Links`**
      - `Input physical ports`
      - `Output physical ports`
      - `Destination`
      - `BAG`
      - J_{max}
      - AC
      - S_{max}
      - `Prioritization`
    - **`Ports`**
      - d_{max}
      - `State`
      - `Speed`
      - `Buffer (low priority)`
      - `Buffer (high priority)`

Table 4 Primitives and parameters for the configurations of the two network protocols HSDN (left) and AFDX (right).

The properties of the AFDX network are described in more detail in the ARINC 664 standard [3]. Because they are not relevant for the remaining work, they are not being discussed in further detail. From the above overview it is apparent that the two protocols HSDN and AFDX deploy agnostic configuration parameters, respectively. One reason is that the AFDX protocol is an implementation of the network technology described in the ARINC 664 standard. HSDN is developed independently with a custom specification for the implementation of the protocol. This becomes particularly evident when observing the characteristics of the two network protocols, as shown in Table 5.

	HSDN	**AFDX**
Global Cycle Time	🟩 Yes	🟥 No
TDMA Schedule	🟩 Strict	🟨 Effective
Time Synchronization	🟩 Synchronous	🟥 No
Buffering	🟨 Cut-through	🟩 Store & forward
Reliable Transmission	🟩 Per-hop	🟨 End-to-end
Authentication	🟩 Yes	🟥 No
Extension	🟩 Optional	🟥 No

Table 5 Comparison of the configuration parameters and properties of the selected network protocols. 🟩 **Fully supported** 🟨 **Undetermined** 🟥 **Not supported**

While HSDN supports *Global Cycle Time*, *Time Synchronization*, *Authentification*, and *Extension*, AFDX does not consider these network functionalities. In contrast, AFDX fully supports buffering of messages between communication participants in a network, while HSDN deploys a cut-through policy.

The comparison of the two selected network protocols makes it apparent that they differ quite significantly from each other. This is contrary to the hypervisor configurations which share a significantly higher set of parameters for configuration even with varying support of the underlying ARINC 653 standard. It is important to understand that the commonalities between network configuration parameters can vary depending on the network protocols used for comparison. However, when observing the differences between two of the more prevalent protocols, which are HSDN and AFDX, we assume sparse commonalities in their configuration. For this reason, a universal avionics configuration should only contain a limited set of common configuration parameters for network protocols. The parameters chosen for a universal avionics network configuration are *Network Participants* – modules and partitions with a network link, *Data Rate* – data rate requirements of the network participants, *Latency* – latency requirements of the network participants, and *Protocol Type* – the type of network protocol to be used in the overall system.

A configuration containing the common parameters presented above, together with a universal hypervisor configuration, are presented in a common configuration in paragraph V.D. Before that, a selection of data serialization formats for capturing configuration parameters is presented in the next subsection.

C. Data Serialization

In the context of software system configurations, data serialization is used for effective management and exchange of information. In avionics, this is particularly evident when referring to standards and development guidances like ARINC 653 that exemplify a hypervisor configuration in the XML format. Data serialization formats enable the structured representation of complex configuration data, facilitating interoperability among diverse systems and components within IMA architectures. By converting data into a standardized, compact form, serialization formats ensure efficient communication and storage of essential avionics system information. Moreover, they enhance the portability and scalability of applications, allowing for seamless integration of new functionalities and adherence to safety-critical requirements. [21, 22] discuss the requirements for avionics configuration elements set by ARINC 653. To extend the two referenced works, the next subsections explore universal avionics configurations with other generic data serialization format.

Serialization Formats

XML is a versatile markup language primarily designed for the storage and transport of data. Characterized by its hierarchical structure, XML allows for the definition of custom tags, enabling a high degree of flexibility in data representation [23]. Its robustness and platform independence makes XML particularly suitable for complex data interchange scenarios, such as web services and document storage. Additionally, XML supports namespaces, which facilitate the management of data from different domains, enhancing interoperability among diverse systems [24].

JSON has emerged as a lightweight data interchange format, recognized for its simplicity and ease of use. Originating from JavaScript, JSON's structure resembles that of objects in the language, making it particularly appealing for web applications [25]. With its concise syntax and readability, JSON has gained widespread adoption in APIs and web services, enabling seamless data exchange between client and server. Furthermore, its language-agnostic nature allows for broad compatibility across various programming environments, positioning JSON as a preferred choice for modern data serialization needs [26].

YAML is a human-readable data serialization format that prioritizes simplicity and clarity. Developed to provide an intuitive alternative to XML and JSON, YAML emphasizes the use of indentation to denote structure, making it particularly accessible for configuration files and data representation. Its ability to express complex data structures in a compact and understandable manner has led to its adoption in a variety of applications, including container orchestration and configuration management tools. The flexibility and ease of editing YAML make it a popular choice among developers and system administrators alike [27].

TOML is a configuration file format designed to be both simple and clear. It seeks to offer an intuitive alternative to more complex formats by maintaining a straightforward syntax that is easy to read and write. TOML's design focuses on clarity, with a clear delineation of data types and structures, making it particularly useful for application configuration. The format's emphasis on human readability, coupled with a comprehensive type system, ensures that it is not only easy to manage but also robust enough for various software applications [28].

Benchmarking

This paragraph aims at quantifying the quality of the different serialization formats introduced above. To perform the quantification, the formats are compared on their performance regarding size, error handling, readability, data support, robustness, third party support, and overall complexity. As shown in [2], these factors play an important role in development, testing, and deployment of such systems. The better the performance of a configuration is, regulated by these metrics, the more efficient and consistent the development process of the overall system can become. Table 6 shows a benchmark of the four data serialization formats XML, JSON, YAML, and TOML quantified with the aforementioned performance metrics.

	XML	**JSON**	**YAML**	**TOML**
Comments	Yes	No	Yes	Yes
Error-Proneness	High	Medium	Medium	Low
Readability	Verbose	Well readable	Human readable	Well readable
Data Structure Support	High	Medium	High	Medium
Indentation Sensitivity	No	No	Yes	No
Third Party Support	High	High	High	Medium
Syntax Complexity	Medium	Low	Medium	Medium
Parsing Support	High	High	Medium	Low

Table 6 Comparison of the data serialization formats XML, JSON, YAML, and TOML [29–33].

Of the four serialization formats, XML is the oldest. While JSON and YAML have a high maturity & third party support in their own right, XML arguably has an even higher one. Also it supports the definition of attributes with an arbitrary depth of child elements for excellent data structure support [24]. The biggest drawbacks of XML as serialization format are its verbosity and syntax complexity. With explicit use of tags and corresponding closing tags, nesting, and obligatory use of root elements, writing configurations in XML can be error-prone and cumbersome. It can also be argued that because of its age and the emergence of newer serialization formats, such as TOML, XML is not as future-proof as other formats.

JSON offers an alternative with similar third party and parsing support as XML. It excels with its good readability and low syntax complexity [26]. However, with the use of brackets and no standard support for comments, the usability and usefulness of JSON can suffer. Also, when comparing the most compact but still readable size of a sample configuration between all formats, JSON by far requires the most lines of code.

As it technically is a superset of it, YAML shares some similarities with JSON. The biggest differences are that YAML does not require brackets, which enhances readability and reduces size, and has a better data structure support with the addition of anchors and date types [27]. While YAML has better data type support, this can come at the cost of syntax complexity. Also, the third party support can be considered lower than XML and JSON.

Last, TOML is a relatively new data serialization format with the aim of making data serialization straightforward. While data structure support is good and syntax complexity is not high, the format suffers from lower maturity, a bigger configuration size, and generally a lower overall support, when compared with the three other data serialization formats [28]. Listings 1 – 4 in the Appendix show comparable instances of ARINC 653-compliant configurations in the XML, JSON, YAML, and TOML data serialization formats, respectively.

Because this research only considers future-proof configuration formats, XML is not further observed. It is debatable which one of the remaining three formats is most suitable for a universal avionics configuration format. For this paper, the authors decided to use YAML for a common configuration, as it supports complex data structures, is concise, and has satisfactory third party and parsing support. The presented serialization formats were not formally tested for their properties and suitability but judged by their specifications from standards and experiences shared by developers. Therefore, a different serialization format than YAML may be applicable for other use cases and needs.

The next subsection discusses an exemplary universal avionics configuration, containing processing as well as network information in a standardized serialization format.

D. Proposed Format for a Unified Avionics System Configuration

The proposed format for a unified avionics system configuration contains all previously explored supersets of parameters for processing modules as well as generic avionics networks found in common avionics configurations. It follows the structure and contains the properties listed in the following tree structure:

NETWORK
- **Type** (enum) – the type of network to be deployed
- **Participants** (int) – the number of participants in the network. This applies to both software defined components as well as hardware modules.
- **Latency** (float) – the maximum latency, in microseconds, defined by system requirements
- **DataRate** (float) – the data rate, in megabits per second, required by the participants

MODULE
- **Name** (String) – the unique name of the module
- **Hypervisor** (String) – the type of hypervisor to be deployed
- **Architecture** (enum) – the CPU architecture of the module which runs the hypervisor
- **BaudRate** (int) – the speed, in signals per second, used for the serial connection with the module which runs the hypervisor
- **BoardOscillator** (float) – the board oscillation of the module in megahertz
- **EntryPoint** (String) – the memory region for the hypervisor image
- **Size** (int) – the reserved size for the hypervisor in bytes
- **CachePolicy** (enum) – the replacement policy for first-level (L1) and second-level (L2) cache
- **ExecutableName** (String) – the name of the hypervisor image
- CPU
  - **Cores** (int) – the number of CPU cores used
  - **Processing** (enum) – the application processing policy. In case of multiple cores being used, symmetric multi-processing (SMP) or asymmetric multi-processing (AMP)
  - **DefaultCore** (int) – the default core to be used for processing
  - **MajorFrame** (int) – the major frame, in nanoseconds, for partitions in a module
- Partitions

↓ *continued on next page* ↓

- Partitions
  - **Name** (String) – the unique name of the partition
  - **ID** (int) – the unique identifier of the partition
  - **PartitionHMNameRef** (String) – the name reference for the partition health monitoring
  - **Duration** (int) – the duration, or partition time window, in nanoseconds, for the execution of the partition
  - **Period** (int) – the periodicity of the partition, in nanoseconds, i.e., the execution frequency per number of major frames
  - **MemoryRegions**
    - **Name** (String) – the unique name of the memory region
    - **Type** (enum) – the type of memory, i.e., random-access memory (RAM), Flash memory, or input/output (I/O) memory
    - **EntryPoint** (String) – the memory region entry point for the partition
    - **Size** (int) – the reserved size of the partition in bytes
    - **CachePolicy** (enum) – the replacement policy for first-level (L1) and second-level (L2) cache
    - **AccessRights** (enum) – the memory access right for the partition, i.e., read, write, or read/write
- Ports
  - **Name** (String) – the unique name of the port
  - **Type** (enum) – the port type, i.e., sampling port or queuing port
  - **EntryPoint** (String) – the memory region entry point for the port
  - **Size** (int) – the reserved size for the port in bytes
  - **MaxMessageSize** (int) – the maximum message size for the port in bytes
  - **MaxNbMessages** (int) – the maximum number of messages that can be buffered by the port. Only applicable for queuing ports, empty for sampling ports.
  - **Destination** (String) – the interface destination partition
- Application
  - **Name** (String) – the unique name of the application
  - **ApplicationReference** (String) – the reference to the application
  - **PartitionReference** (String) – the reference to the partition to which the application is mapped
- Schedules
  - **ID** (int) – the unique identifier of the schedule
  - **CPU** (int) – the reference to the CPU core this schedule applies to
  - **PartitionWindow**
    - **PartitionReference** (String) – the reference to the partition for which this partition time window is defined
    - **PeriodicProcessingStart** (bool) – periodic processing of the partition
    - **Offset** (int) – the offset, in nanoseconds, for the partition time window
    - **Duration** (int) – the execution duration, in nanoseconds, of the partition time window
- HealthMonitoring
  - **Name** (String) – the unique name of the health monitoring entry
  - **NbHmLogs** (int) – the number of health monitoring logs
  - **Event**
    - **Name** (String) – the unique name of the health monitoring event
    - **ID** (int) – the unique identifier of the health monitoring event
    - **Description** (String) – the description of the health monitoring event
    - **Action** (String) – the required action in case the health monitoring event is triggered
    - **EventReference** (String) – the reference to another, related, event for further action

The configuration is captured in a single file with one root element for the universal network properties and one root element for the universal module properties. A complete example in YAML format is presented in Listing 5.

The proposed configuration is a first sketch for a universal avionics configuration. The authors made a few assumptions during development that lead to following constraints. First, the configuration may not be complete as only a limited selection of hypervisors was considered. At the same time, the configuration could contain redundant information, depending on the specific end system used for deployment. This is down to the the universal nature of the presented approach. The superset of information will be redundant to some degree for end systems that do not support all functionalities contained in the universal configuration. Last, some of the information might lead to incompatibilities depending on different use cases and integrations.

Nevertheless, with the proposed universal configuration, the authors present a first pragmatic concept for unifying avionics system configuration which is planned to evolve with future work.

VI. Discussion

The findings of this research underscore the necessity for a unified avionics system configuration that addresses the complexities and inconsistencies currently faced in the avionics industry. The identification of common properties across various avionics systems serves as a critical step toward achieving this goal, as it allows for the establishment of a standardized framework that can accommodate diverse computing modules and network protocols. The study evaluates various serialization formats and concludes that YAML is among the most suitable choices for a universal avionics configuration, given its ability to manage complex data structures while remaining concise and user-friendly. By establishing a superset of essential avionics attributes, the paper facilitates improved interoperability and coherence in system configurations, ultimately enhancing the flexibility and efficiency of the engineering process.

However, several limitations must be acknowledged. First, the proposed universal configuration format is based on a theoretical framework that requires validation through real-world applications. While the research provides a comprehensive overview of the essential attributes and serialization formats, the practical implementation of the configuration format in existing avionics systems remains to be explored. Future studies should focus on case studies that demonstrate the effectiveness in various operational environments. Additionally, the choice of YAML as the preferred serialization format, while justified by its advantages in handling complex data structures, may not be universally applicable across different system environments. Different systems may have unique requirements that necessitate the use of alternative formats.

In terms of future extensions, the development of tools and frameworks that support the development of a unified avionics configuration would be beneficial. These tools could automate the configuration process, ensuring consistency and reducing the likelihood of errors. One of those tools is the SysML v2 language which is currently in its final stage of specification to be released in the coming months. SysML v2 adds textual notation to systems modeling which bears powerful potential for the aforementioned extensions, such as automated development of avionics configurations and formalized validation. Moreover, the use of ontologies could enhance the work at hand. Ontologies provide a structured framework for knowledge representation and relationships among components. They facilitate efficient data exchange through semantic interoperability, support automated reasoning for configuration validation, and allow for easy adaptation to evolving technologies. Additionally, they assist in informed decision-making by offering a comprehensive view of component relationships, ultimately contributing to more efficient and reliable systems. The use of SysML v2 as a Model-Driven Development tool with the addition of ontologies will be explored in future publications.

VII. Conclusion

In this paper, we presented the challenges and opportunities associated with avionics system configurations. The current landscape reveals a significant gap in the standardization of configuration formats, which often leads to inefficiencies and inconsistencies in the development and integration of avionics systems. To address these challenges, we proposed an avionics configuration which captures essential elements, from computing resources to network protocols, in a universal and standardized format.

Our research highlights the importance of establishing a superset of essential avionics attributes that can serve as a foundation for further avionics system development. By leveraging a serialization format that supports complex data structures, such as YAML, we aim to enhance the flexibility and coherence of the engineering process in avionics development. The proposed universal avionics configuration not only facilitates better interoperability among diverse components but also streamlines the configuration process, ultimately contributing to more efficient and reliable avionics systems.

While the findings of this study provide a solid framework for future advancements in avionics configurations, it is essential to recognize the limitations and challenges associated with implementing a standardized format across the industry. Future research should focus on empirical validation of a universal configuration format in real-world applications, as well as exploring the adaptability of the proposed format to various avionics systems and technologies.

References

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

[2] 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)*, 2024, pp. 1–10. https://doi.org/10.1109/DASC62030.2024.10749454.

[3] ARINC, "AIRCRAFT DATA NETWORK AVIONICS FULL-DUPLEX SWITCHED ETHERNET NETWORK," Standard, Aeronautical Radio Incorporated, Annapolis, MD, USA, Feb. 2006.

[4] 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.

[5] Chrysalidis, P., and Thielecke, F., "Universal Configuration Format for Virtual, Hybrid and Hardware Testing of Avionic Platforms," *2024 IEEE/AIAA 43rd Digital Avionics Systems Conference (DASC)*, 2024, pp. 1–10. Submitted for publication.

[6] Eriksson, M., and Hallberg, V., "Comparison between JSON and YAML for Data Serialization." 2011. URL https://api.semanticscholar.org/CorpusID:60310679.

[7] Sumaray, A., and Makki, S. K., "A comparison of data serialization formats for optimal efficiency on a mobile platform," *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, Association for Computing Machinery, New York, NY, USA, 2012. https://doi.org/10.1145/2184751.2184810.

[8] Kaur, A., Ayyagari, S., Mishra, M., and Thukral, R., "A Literature Review on Device-to-Device Data Exchange Formats for IoT Applications," *JOURNAL OF INTELLIGENT SYSTEMS AND COMPUTING*, Vol. 1, No. 1, 2020, pp. 1—-10. URL http://scienceandtech.co.uk/journals/index.php/jiscom/article/view/4.

[9] Truică, C.-O., Apostol, E.-S., Darmont, J., and Pedersen, T. B., "The Forgotten Document-Oriented Database Management Systems: An Overview and Benchmark of Native XML DODBMSes in Comparison with JSON DODBMSes," *Big Data Res.*, Vol. 25, No. C, 2021. https://doi.org/10.1016/j.bdr.2021.100205.

[10] Papagiannopoulos, I., "JSON application programming interface for discrete event simulation data exchange," Master's thesis, University of Limerick, Limerick, Jan. 2015. Available at https://researchrepository.ul.ie/articles/thesis/JSON_application_programming_interface_for_discrete_event_simulation_data_exchange/19840702.

[11] Xia, S., and Chen, L., "HighYaml: A High-Performance YAML Serialization and Deserialization Framework for Python," *Elsevier*, 2024. https://doi.org/10.2139/ssrn.4978370, preprint, submitted for publication.

[12] Intel Corporation, "Delivering Certified, Safety-Critical Computing for the Future of Aviation," Tech. rep., Intel Corporation, Jul. 2023. URL https://www.intel.co.id/content/dam/www/central-libraries/us/en/documents/ddci-deos-rtos-solution-brief.pdf.

[13] SYSGO GmbH, "PikeOS Certifiable RTOS with Hypervisor Functionality," Tech. rep., SYSGO GmbH, Jul. 2024. URL https://www.sysgo.com/fileadmin/user_upload/data/flyers_brochures/SYSGO_PikeOS_Product_Note.pdf.

[14] Burns, A., and Davis, R. I., *Mixed Criticality Systems - A Review (13th Edition)*, White Rose, 2022. URL https://eprints.whiterose.ac.uk/183619/.

[15] Masmano, M., Ripoll, I., Crespo, A., and Jean-Jacques, M., "XtratuM: a Hypervisor for Safety Critical Embedded Systems." *11th Real-Time Linux Workshop*, Instituto de Informática Industrial and Universidad Politécnica de Valencia, Spain and CNES France, 2009.

[16] Crespo, A., Ripoll, I., Masmano, M., Arberet, P., and Jean-Jacques, M., "XtratuM: An Open Source Hypervisor for TSP Embedded Systems in Aerospace," 2009, pp. 31–38.

[17] Schuster, T., and Verma, D., "Networking concepts comparison for avionics architecture," *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, 2008, pp. 1.D.1–1 – 1.D.1–11. https://doi.org/10.1109/DASC.2008.4702761.

[18] Alena, R. L., Ossenfort, J. P., Laws, K. I., Goforth, A., and Figueroa, F., "Communications for Integrated Modular Avionics," *2007 IEEE Aerospace Conference*, 2007, pp. 1–18. https://doi.org/10.1109/AERO.2007.352639.

[19] Plankl, H., "MIL-STD-1553B and it's potential for the future," *The European Test and Telemetry Conference*, 2018, pp. 114–122. https://doi.org/10.5162/ettc2018/6.1.

[20] Schmidt, R., Cherian, S., Münz, F., Engel, H., and Reubold, T., "A Deterministic, Versatile and Time-Synchronized Data Network Infrastructure for Next Generation Modular Avionics," *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024, pp. 1–10. https://doi.org/10.1109/DASC62030.2024.10748899.

[21] Lukić, B., Friedrich, S., Schubert, T., and Durak, U., "Automated Configuration of ARINC 653-Compliant Avionics Architectures," *AIAA SCITECH 2024 Forum*, 2024, pp. 1–17. https://doi.org/10.2514/6.2024-1856.

[22] Lukić, B., Friedrich, S., and Durak, U., "A Streamlined Approach Toward Automated Generation and Validation of ARINC 653–Compliant Avionics Configurations," *Journal of Aerospace Information Systems*, Vol. 22, No. 5, 2025, pp. 314–324. https://doi.org/10.2514/1.I011439.

[23] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F., *Extensible Markup Language (XML) 1.0*, W3C, Wakefield, MA, USA, fifth edition ed., Nov. 2008. URL https://www.w3.org/TR/xml/.

[24] Nolan, D., and Lang, D. T., *An Introduction to XML*, Springer New York, New York, NY, 2014, pp. 19–52. https://doi.org/10.1007/978-1-4614-7900-0_2.

[25] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)," RFC 4627, Jul. 2006. https://doi.org/10.17487/RFC4627.

[26] Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoč, D., "Foundations of JSON Schema," *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2016, pp. 263–273. https://doi.org/10.1145/2872427.2883029.

[27] Ben-Kiki, O., Evans, C., and Ingerson, B., "YAML Ain't Markup Language (YAML) (tm) Version 1.2," Tech. rep., YAML.org, Sep. 2009. URL http://www.yaml.org/spec/1.2/spec.html.

[28] Preston-Werner, T., and Gedam, P., "TOML Specification, Version 1.0.0-rc.1," Tech. rep., TOML.io, Apr. 2020. URL https://toml.io/en/v1.0.0-rc.1.

[29] Voss, A., and Lucas, T., "JSON Schema Everywhere," https://json-schema-everywhere.github.io/, json-schema-everywhere.github.io, Nov. 2024. [Accessed 30-11-2024].

[30] Nicolas S., "Parsing JSON is a Minefield," https://seriot.ch/projects/parsing_json.html, seriot.ch, Mar. 2018. [Accessed 30-11-2024].

[31] Hunter, G., "A Comparison Of Serialization Formats," https://blog.mbedded.ninja/programming/serialization-formats/a-comparison-of-serialization-formats, mbedded.ninja, May 2019. [Accessed 11-11-2024].

[32] Ueding, M., "JSON vs. YAML vs. TOML," https://martin-ueding.de/posts/json-vs-yaml-vs-toml, martin-ueding.de, Aug. 2022. [Accessed 11-11-2024].

[33] Parakin, D., "Config Files: INI, XML, JSON, YAML, TOML," https://www.barenakedcoder.com/blog/2020/03/config-files-ini-xml-json-yaml-toml/#json, barenakedcoder.com, Mar. 2020. [Accessed 11-11-2024].

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.com/aeec/ARINC653/
    P1S5 ExampleSchema.xsd" Name="MyModule">
   <Partitions>
      <Partition>
         <PartitionDefinition Name="TAWS" Identifier="1"
             PartitionHMNameRef="Terrain Avoidance and Warning System HM table"/>
         <PartitionPeriodicity Duration="50000000" 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>
         </PartitionPorts>
      </Partition>
   </Partitions>
   <Schedules>
      <PartitionTimeWindow PeriodicProcessingStart="true" Duration="50000000"
          PartitionNameRef="TAWS" Offset="0"/>
      <PartitionTimeWindow PeriodicProcessingStart="true" Duration="25000000"
          PartitionNameRef="TAWS" Offset="50000000"/>
   </Schedules>
   <HealthMonitoring>
      <SystemErrors>
         <SystemError ErrorIdentifier="1"
             Description="Configuration Error"/>
         <SystemError ErrorIdentifier="2"
             Description="Module Config Error"/>
         <SystemError ErrorIdentifier="3"
             Description="partition config error"/>
      </SystemErrors>
   </HealthMonitoring>
</MODULE>
```

Listing 1 ARINC 653-compliant sample configuration in XML format.

```json
"MODULE": {
   "Partitions": {
      "Partition": [
         {
            "PartitionDefinition": {
               "@_Name": "TAWS",
               "@_Identifier": "1",
               "@_PartitionHMNameRef": "Terrain Avoidance and
                   Warning System HM table"
            },
            "PartitionPeriodicity": {
               "@_Duration": "50000000",
               "@_Period": "100000000"
            },
            "MemoryRegions": {
               "MemoryRegion": [
                  {
                     "@_Type": "RAM",
                     "@_Size": "1048576",
                     "@_Name": "mainMemory",
                     "@_AccessRights": "READ_WRITE"
                  },
                  {
                     "@_Type": "Flash",
                     "@_Size": "524288",
                     "@_Name": "Flash",
                     "@_AccessRights": "READ_ONLY"
                  }
               ]
            },
            "PartitionPorts": ""
         }
      ]
   },
   "Schedules": {
      "PartitionTimeWindow": [
         {
            "@_PeriodicProcessingStart": "true",
            "@_Duration": "50000000",
            "@_PartitionNameRef": "TAWS",
            "@_Offset": "0"
         },
         ...
         ...
      ]
   },
   ...
   ...
}
```

Listing 2 ARINC 653-compliant sample configuration in JSON format.

```yaml
MODULE:
    Partitions:
        Partition:
            -
                PartitionDefinition:
                    '@_Name': TAWS
                    '@_Identifier': '1'
                    '@_PartitionHMNameRef': Terrain Avoidance and
                        Warning System HM table
                PartitionPeriodicity:
                    '@_Duration': '50000000'
                    '@_Period': '100000000'
                MemoryRegions:
                    MemoryRegion:
                        -
                            '@_Type': RAM
                            '@_Size': '1048576'
                            '@_Name': mainMemory
                            '@_AccessRights': READ_WRITE
                        -
                            '@_Type': Flash
                            '@_Size': '524288'
                            '@_Name': Flash
                            '@_AccessRights': READ_ONLY
                PartitionPorts: ''
    Schedules:
        PartitionTimeWindow:
            -
                '@_PeriodicProcessingStart': 'true'
                '@_Duration': '50000000'
                '@_PartitionNameRef': TAWS
                '@_Offset': '0'
            -
                '@_PeriodicProcessingStart': 'true'
                '@_Duration': '25000000'
                '@_PartitionNameRef': TLS-ADSB
                '@_Offset': '50000000'
    HealthMonitoring:
        SystemErrors:
            SystemError:
                -
                    '@_ErrorIdentifier': '1'
                    '@_Description': Configuration Error
                -
                    '@_ErrorIdentifier': '2'
                    '@_Description': Module Config Error
                -
                    '@_ErrorIdentifier': '3'
                    '@_Description': partition config error
```

Listing 3 ARINC 653-compliant sample configuration in YAML format.

```
[MODULE]

[[MODULE.Partitions.Partition]]
PartitionPorts = ""

    [MODULE.Partitions.Partition.PartitionDefinition]
    "@_Name" = "TAWS"
    "@_Identifier" = "1"
    "@_PartitionHMNameRef" = "Terrain Avoidance and Warning
        System HM table"

    [MODULE.Partitions.Partition.PartitionPeriodicity]
    "@_Duration" = "50000000"
    "@_Period" = "100000000"

[[MODULE.Partitions.Partition.MemoryRegions.MemoryRegion]]
"@_Type" = "RAM"
"@_Size" = "1048576"
"@_Name" = "mainMemory"
"@_AccessRights" = "READ_WRITE"

[[MODULE.Partitions.Partition.MemoryRegions.MemoryRegion]]
"@_Type" = "Flash"
"@_Size" = "524288"
"@_Name" = "Flash"
"@_AccessRights" = "READ_ONLY"

[[MODULE.Schedules.PartitionTimeWindow]]
"@_PeriodicProcessingStart" = "true"
"@_Duration" = "50000000"
"@_PartitionNameRef" = "TAWS"
"@_Offset" = "0"

[[MODULE.Schedules.PartitionTimeWindow]]
"@_PeriodicProcessingStart" = "true"
"@_Duration" = "25000000"
"@_PartitionNameRef" = "TAWS"
"@_Offset" = "50000000"

[[MODULE.HealthMonitoring.SystemErrors.SystemError]]
"@_ErrorIdentifier" = "1"
"@_Description" = "Configuration Error"

[[MODULE.HealthMonitoring.SystemErrors.SystemError]]
"@_ErrorIdentifier" = "2"
"@_Description" = "Module Config Error"

[[MODULE.HealthMonitoring.SystemErrors.SystemError]]
"@_ErrorIdentifier" = "3"
"@_Description" = "partition config error"
```

Listing 4 ARINC 653-compliant sample configuration in TOML format.

```
NETWORK:
    '@_Type': HSDN
    '@_Participants': '1'
    '@_Latency': '80'
    '@_DataRate': '100000'


MODULE:
    '@_Name': myModule
    '@_Hypervisor': pikeOS
    '@_Architecture': ppc_e500mc_4g
    '@_BaudRate': '115200'
    '@_BoardOscillator': '50'
    '@_EntryPoint': 0x1000000
    '@_Size': '1000000'
    '@_CachePolicy': l1WriteBack_l2WriteBack
    '@_ExecutableName': image.bin

    CPU:
        '@_Cores': '1'
        '@_Processing': none
        '@_DefaultCore': '0'
        '@_MajorFrame': '100000000'

    Partitions:
        -
            '@_Name': partition1
            '@_ID': '1'
            '@_PartitionHMNameRef': partition1HMTable
            '@_Duration': '50000000'
            '@_Period': '100000000'
            MemoryRegions:
                -
                    '@_Name': mainMemory
                    '@_Type': RAM
                    '@_EntryPoint': 0x8000000
                    '@_Size': '1000000'
                    '@_CachePolicy': l1WriteBack_l2WriteBack
                    '@_AccessRights': readWrite

        -
            '@_Name': partition2
            '@_ID': '2'
            '@_PartitionHMNameRef': partition2HMTable
            '@_Duration': '50000000'
            '@_Period': '100000000'
            MemoryRegions:
                -
                    '@_Name': mainMemory
                    '@_Type': Flash
                    '@_EntryPoint': 0x8000000
                    '@_Size': '4000000'
                    '@_CachePolicy': l1WriteBack_l2WriteBack
                    '@_AccessRights': readWrite

    Ports:
        -
            '@_Name': port1
            '@_Type': Queuing
            '@_EntryPoint': 0x400000
            '@_Size': '10000'
            '@_MaxMessageSize': '1000'
            '@_MaxNbMessages': '10'
            '@_Source': partition1
            '@_Destination': partition2

    Application:
        -
            '@_Name': avionicsApplication
            '@_ApplicationReference': /avionics_application.elf
            '@_PartitionReference': partition1

    Schedules:
        -
            '@_ID': '1'
            '@_CPU': '0'
            PartitionWindow:
                -
                    '@_PartitionReference': partition1
                    '@_PeriodicProcessingStart': 'true'
                    '@_Offset': '0'
                    '@_Duration': '50000000'
                -
                    '@_PartitionReference': partition2
                    '@_PeriodicProcessingStart': 'true'
                    '@_Offset': '50000000'
                    '@_Duration': '50000000'

    HealthMonitoring:
        -
            '@_Name': Health Monitoring Table
            '@_NbHmLogs': '64'
            Event:
                -
                    '@_Name': numericError
                    '@_ID': '0'
                    '@_Description': A description for the numeric error event
                    '@_Action': haltPartition
                    '@_EventReference': none
```

Listing 5 Universal avionics system configuration containing network and computing resource properties in YAML format.