





Exploring SysML v2 for Model-Based Engineering of Safety-Critical Avionics Systems

Alexander Ahlbrecht 
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
alexander.ahlbrecht@dlr.de

Wanja Zaeske 
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
wanja.zaeske@dlr.de

Bojan Lukić 
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
bojan.lukic@dlr.de

Umut Durak 
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
umut.durak@dlr.de

Abstract—For complex and safety-critical avionics systems, distributed developments are unavoidable. To coordinate the fragmented development process, a seamless information exchange is necessary. A promising paradigm to coordinate the development activities is Model-Based Systems Engineering (MBSE). However, exchanging information with current MBSE practices still presents interoperability challenges. The second version of the Systems Modeling Language (SysML v2) introduces potential solutions to these challenges. A key feature is the standardized Application Programming Interface (API), presenting an opportunity for server-based data exchanges. Since it is not yet explored how the SysML v2 features can be utilized for the development of avionics systems, this paper provides an overview. Therefore, a SysML v2 avionics application concept is outlined, exercised, and discussed. Overall, SysML v2 shows potential to complement model-based avionics practices through novel features and the standardized API. Still, domain-specific adjustments are necessary to apply the general-purpose language in the avionics context.

Index Terms—Avionics, MBSE, SysML v2, API, CI/CD

I. INTRODUCTION

The rapid advancement of avionics technology has led to an exponential increase in system complexity with increasing numbers of software-defined functions and interconnected components. As a result, the traditional methods for designing, testing, and certifying avionics systems are struggling to keep pace [1]. Particularly, the fragmented development environments are leading to inefficiencies, errors, and inconsistencies during the complex system developments [2].

A paradigm to address the issue of fragmented developments is Model-Based Systems Engineering (MBSE). It promises to connect multiple development stages by applying a model-based and therefore semi-formal format for storing and exchanging information [3]. Such a model-based information exchange is especially interesting for the development of complex systems, where model-based approaches are applied at every development level. In practice, however, a seamless integration of model-based development activities is often not

achieved. Reasons are the different model-based representations at every development stage as well as the limited standardization, acceptance, and tool implementation of exchange formats. As a result, the transition of novel methods from academia to industry is challenging [4].

To tackle the aforementioned challenges, a standardized and widely accepted way of storing and exchanging model information is needed. In this context, the second version of the Systems Modeling Language (SysML v2) has the potential to contribute as a standardized, precise, expressive, and, most importantly, interoperable modeling language. However, it is not yet explored how SysML v2 can be applied in the context of safety-critical avionics developments. Accordingly, this paper contributes by:

- Highlighting key features of SysML v2
- Introducing a SysML v2 avionics application concept
- Demonstrating the concept with an example
- Discussing domain-specific areas of improvement

The paper is structured in the following manner. Initially, some background information about the current avionics engineering practices and the SysML v2 are provided in Section II. Section III presents related work and current limitations. Then, a concept is proposed outlining how SysML v2 can be applied in the context of model-based avionics engineering in Section IV. Afterwards, the proposed concept is demonstrated in a small use case in Section V. Finally, the opportunities and limitations of SysML v2 are discussed in Section VI and the paper is concluded in Section VII.

II. BACKGROUND

A. Avionics Engineering

The development of avionics systems has to conform to strict standards and guidelines. Following the standards and guidelines helps to generate a suitable, safe, and certifiable avionics system. Since the use case in this paper will cover the

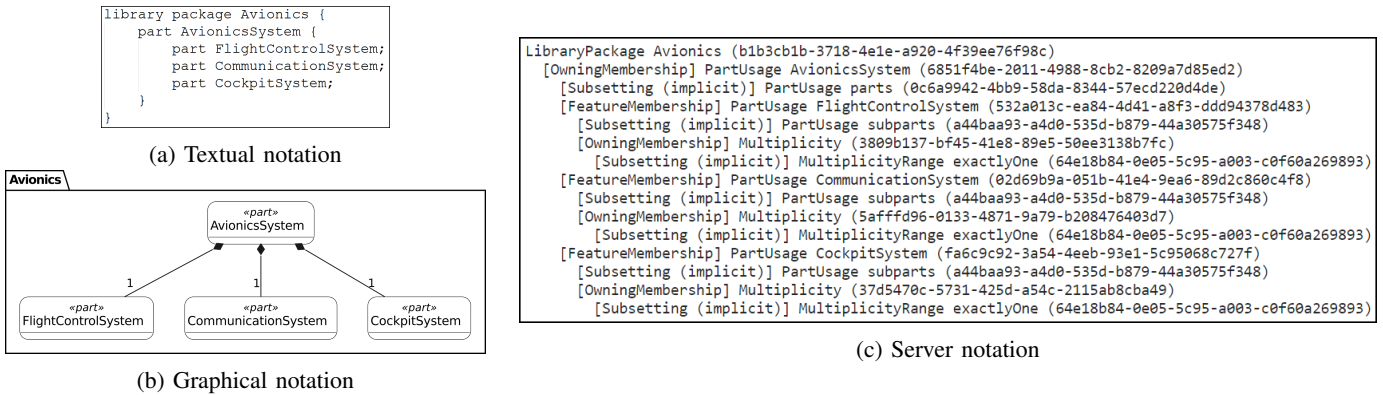


Fig. 1: Different SysML v2 representations of the same system model

model-based configuration of an Integrated Modular Avionics (IMA) system, the most important standards for the context of this paper are the DO-297 (IMA Development Guidance and Certification Considerations) [5], DO-330 (Software Tool Qualification Considerations) [6], and the ARINC 653 (Avionics Application Software Standard Interface) [7].

IMA is an innovative architecture principle allowing a modular integration of applications onto a shared computing platform. In IMA systems, modularity is a key principle for hardware and software [2]. This principle leads to improved system integration, reliability, and maintenance properties [8]. A common way of implementing the software application interface on an IMA platform are ARINC 653 partitioned environments [9]. By adhering to ARINC 653, it is ensured that different software modules, known as partitions, operate independently and do not interfere with each other [10].

During the development of IMA systems, model-based approaches are applied at multiple stages of the development. First, a high-level model is established by describing requirements, structure, behavior, and parameters of the system. At this stage, initial architecture decisions are made (e.g. using trade-off analyses). For the implementation phase, the high-level architecture models are iteratively refined. In parallel, parameters and behavior are modeled and used for verification and validation (e.g. simulation, model-checking). Ultimately, a final representation of the systems is created, implemented, and deployed on the avionics hardware [8].

For each development stage, dedicated modeling techniques and domain-specific languages exist. Even though there are many attempts to improve the connection and consistency across the development stages, the lack of sufficient standardization complicates the interoperability. One reason is the varying formal bases of the models, both at the syntactic and semantic level. Furthermore, each tool vendor implements the storing and access to model information in a different way, reducing the interoperability even further.

B. SysML v2

SysML v2 is a general-purpose modeling language enabling the specification, analysis, design, and verification of complex

systems [11]. The language supports to model requirements, structure, behavior, and parameters of the system. Simultaneously, domain-specific extensions are enabled with the library concept. In comparison to SysML v1, SysML v2 is implemented in the newly developed Kernel Modeling Language (KerML). By building on top of KerML, novel features such as textual notation are enabled. The textual notation can be used in parallel to the graphical notation as shown in Fig. 1.

Overall, SysML v2 addresses various limitations and challenges of SysML v1. Improvements aim at enhancing usability, expressiveness, and integration capabilities [12], [13]. The usability is improved with aspects such as refined language constructs and more consistent terminology. In terms of enhanced expressiveness, the language now supports the modeling of system variants, analysis cases, verification cases, and views. Finally, for improved integration capabilities, a standardized Application Programming Interface (API) was defined for the SysML v2 [14]. The standardized API and related concept of a central model server introduce new opportunities for the application of the language.

A generic application concept for the language is presented in Fig. 2a. In this concept, the model server and standardized API are not only allowing to improve the exchange between dedicated SysML v2 tools but also enable to connect domain tools or programming languages. Such a standardized exchange also facilitates the development of tools that only focus on a specific aspect of the system development process. Thereby, opportunities are created for innovative tools by small companies, universities, or open-source communities. Simultaneously, the established industry vendors show interest in supporting the SysML v2, due to the advanced capabilities and interoperability ambitions of the language. To get started with the SysML v2, material such as [15] as well as the documentation at the SysML-v2-GitHub¹ are recommended.

III. RELATED WORK

Since the goal of this paper is to highlight model-based development and integration possibilities for avionics systems, the related work will focus on similar and complementary

¹SysML v2: <https://github.com/Systems-Modeling/SysML-v2-Release>

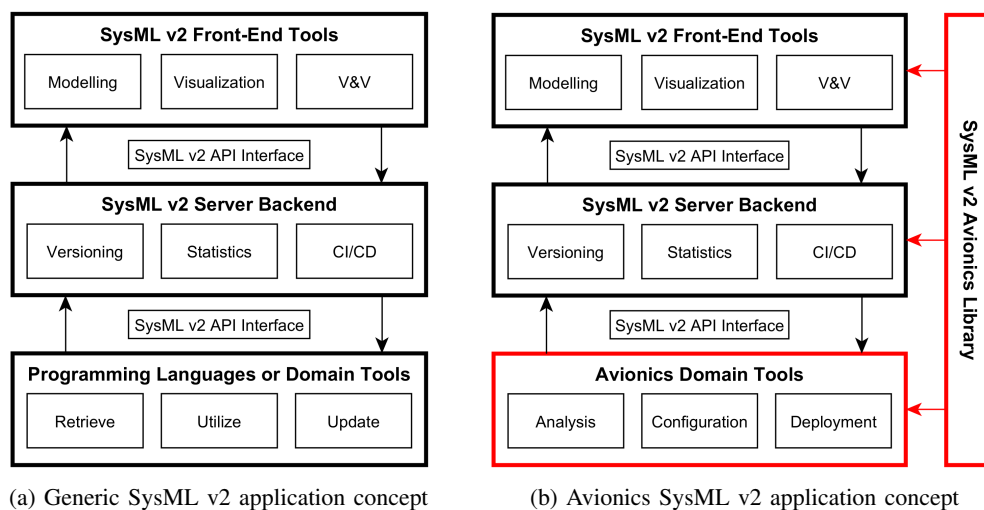


Fig. 2: Comparison of generic and avionics specific SysML v2 application concepts

approaches. For the model-based development of avionics systems, Domain-Specific Languages (DSLs) are a promising solution. A DSL in the avionics domain is the Open Avionics Architecture Model (OAAM) [16]. OAAM can be used to create, manage, and configure avionics models. Creating OAAM models is supported by tools [17], a dedicated query language [18], and tool qualification concepts [19].

In addition to DSLs, general-purpose modeling languages can be used and extended to model avionics systems. One example is the Architecture Analysis & Design Language (AADL). For instance, [20] shows how the ARINC 653 software concepts can be modeled and validated using AADL. Since the SysML v2 is also a general-purpose modeling language, some similarities to the AADL exist and related work highlights the integration and transformation opportunities between the languages [21], [22].

When the avionics system is represented in a model, the model can be used to support domain-specific activities such as the system and software configuration [10], [23], [24], network optimization [25], [26], or simulation [27]. At the same time, model-based approaches facilitate the overarching system design process as outlined in [28] as well as the AvioNET [29], [30] and eSAM approaches [31].

Considering the related work, it is apparent that multiple domain-specific tools and languages exist for the model-based development of avionics systems. It is also noteworthy that connecting the development activities is of interest and concepts exist in the related work. However, current solutions are often not standardized and require dedicated connector applications to industry tools. These connectors often implement only a subset of the DSLs, and are prone to breakage on software updates. After all, they attempt to create interoperability, where it was not an explicit design goal of the DSLs in the first hand. The goal would be a standardized exchange between all relevant industry and open-source tools. For this goal, the API of SysML v2 could be very relevant.

IV. SYSML v2 AVIONICS APPLICATION CONCEPT

After showcasing related work in the previous section, this section will cover how the SysML v2 could be applied in the context of model-based avionics engineering. A SysML v2 driven avionics engineering concept is highlighted in Fig. 2b. The four main components are the *SysML v2 Avionics Library*, the *SysML v2 Front-End Tools*, the *SysML v2 Server Backend*, and the *Avionics Domain Tools*. By establishing a *SysML v2 Avionics Library*, the domain-specific concepts are integrated into the general-purpose language. As a result, the *SysML v2 Front-Ends* are able to model, visualize, and check all relevant aspects of the avionics system. In parallel, the avionics model is serialized and stored in the *SysML v2 Server Backend*. Finally, the server model is accessed by *Avionics Domain Tools* to assist in tasks of the avionics development. In the following, a more detailed description of each component is provided.

A. *SysML v2 Avionics Library*

For the development of an avionics system various aspects have to be designed, implemented, and configured. First, requirements must be collected and used to design the system architecture. Then, the system architecture has to be refined, configured, and tested. On the higher architecture levels, functionality is collected and allocated to different devices. At the same time, the data exchange is defined and specific solution technologies are selected. Then, avionics-specific properties are configured for each device such as the runtime environment (operating system, etc.), interfaces (software, hardware, etc.), resources (memory, scheduling, etc.), and monitors (health, safety, security, etc.).

Since the SysML v2 is a capable general-purpose language, all of these aspects can be represented. However, the application benefits from a domain-specific adjustment of the language to systematically represent the relevant avionics elements and properties. The SysML v2 supports the extension with domain-specific libraries. For instance, a domain-specific

SysML v2 library for the OAAM [16] DSL could be created as demonstrated for the OAAM HardwareLayer in Fig. 3.

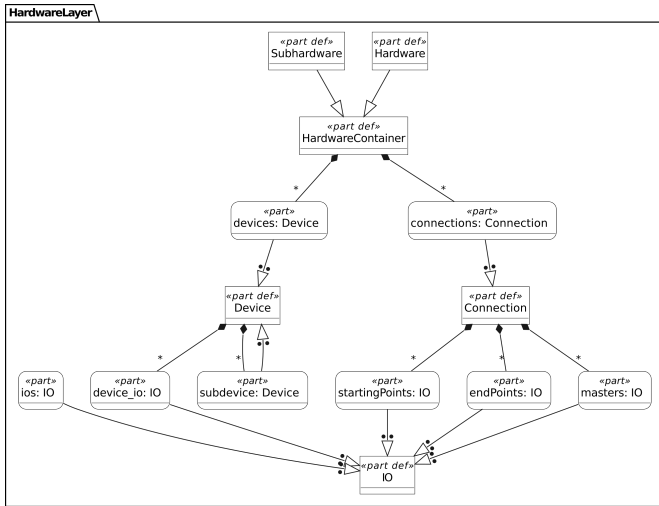


Fig. 3: HardwareLayer of OAAM [16] as SysML v2 library

B. SysML v2 Front-End Tools

By using dedicated SysML v2 tools in combination with a domain-specific library, a SysML v2 conform avionics model can be created, visualized, and checked. Since typical industry vendors of MBSE tools are looking to support the SysML v2, their tools will fall under this category. Such MBSE tools bring capabilities to create and visualize models, execute design trade-offs, check the conformity of models, etc. Since the SysML v2 language was extended in usability and expressiveness as described in Section II-B, these advantages will also be present in dedicated SysML v2 tools. During the creation of the standard, a proof-of-concept implementation was set up to enable to try out the language. In this paper, this implementation was used to create the SysML v2 visualizations. An example can be seen in Fig. 3.

Moreover, in contrast to current solutions, the model creation and visualization should not be limited to one specific tool. Instead, the idea is that multiple tools can contribute and exchange models via a SysML v2 server. This could also help to ease the distributed working that is common during complex avionics developments. For the development of IMA systems, this could help to respect the different roles defined in DO-297 [5] where each involved company can use their preferred tooling. Moreover, avionics-specific SysML v2 tools can be developed and integrated with complex industry tools.

C. SysML v2 Server Backend

A central component to enable the distributed working and tool interoperability are SysML v2 servers and their standardized API. They are used to store all relevant model information and keep the model updated across multiple tools. Therefore, a git-like commit process can be utilized. Such a server concept also enables other aspects that are very relevant for avionics developments. Namely, the development

can branch out, where different features of the design are developed in separate branches. Once ready, the branches can be merged.

Another server concept is versioning. Versioning helps to achieve systematic configuration management which is demanded by typical aviation standards. Moreover, different roles could be established for the different companies involved in the avionics development to manage controlled read and write access. In this regard, properties such as privacy of data would also have to be considered and managed. For instance, the server could store the data in an encrypted form and only provide the data to the entitled development participants.

Another approach facilitated by a central server with git-like commits is Continuous Integration (CI) and Continuous Deployment (CD). CI/CD can be used to automatically check and merge changes (CI) and deploy the resulting/derived artifacts (CD). In the context of avionics, CI/CD pipelines could be set up to check architecture designs, generate configurations, or automatically execute a deployment. An example pipeline is shown in Fig. 4. After a model commit, the consistency of the provided information is checked. When the consistency check was successful, the architecture optimization is executed and updates the model parameters where required. Then, the configuration is generated. A final check confirms the suitability of the configuration, deploys the configuration, and generates a final report. The CI/CD approach also supports the collection of statistics for commits, test cases, and other aspects. Upon failure in one of the pipeline steps, detailed feedback can be provided immediately.

D. Avionics Domain Tools

Since the main goal of this paper is to tackle the fragmented avionics developments, the integration of avionics domain tools is a key component of the SysML v2 avionics concept. In the related work described in Section III, some avionics-specific tools were introduced for activities such as network optimization [25], system and software configuration [23], [24], or analysis via simulation [27].

In the proposed concept, domain-specific tools are connected via the standardized API. Required model information is queried and resulting updates in the models are committed to the server. Because the standardized API can be accessed easily with e.g. REST/HTTP requests, the domain-specific tools can be implemented with any programming language of choice. Overall, the process of connecting domain-specific tools to a central SysML v2 server enables to keep the specific advantage of each domain-specific tool while establishing an overarching and consistent system view.

V. SysML v2 AVIONICS CONCEPT DEMONSTRATION

Since the demonstration of all aspects of the concept for the whole avionics development is not feasible in one paper, this paper focuses on demonstrating the concept with a part of the avionics development. Concretely, the focus is on the configuration of software partitions according to the ARINC 653 standard. For demonstration purposes, an open-source

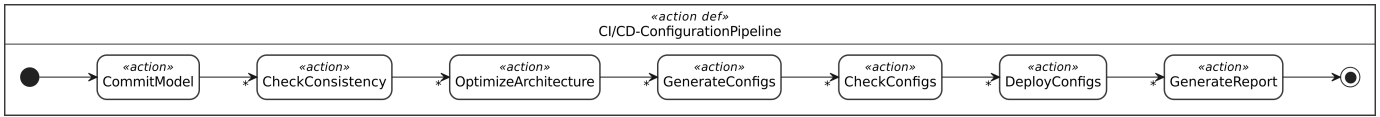


Fig. 4: CI/CD pipeline example for configuration generation

ARINC 653 Linux hypervisor² will be used. The goal in this demonstration is to model and configure the server-client ping example of the ARINC 653 Linux hypervisor.

A. ARINC 653 Hypervisor Library

To be able to model the specific aspects of the Linux hypervisor, a corresponding SysML v2 library was created. The library is displayed in the textual notation of SysML v2 in Fig. 5. Initially, the *import* keyword is applied to define the *String* and *Integer* types for the attributes. Then, the main components (parts) are defined which are *Hypervisor* and *Partition*. Each hypervisor can have multiple partitions while both have timing attributes. In addition, ports are also defined. The two specializations (*:>*) of the *PartitionPort* type are the *QueuingPort* and *SamplingPort*. Both port types inherit the *msg_size* attribute.

```

1. library package LinuxHypervisorLibrary {
2.
3.   import ScalarValues::String;
4.   import ScalarValues::Integer;
5.
6.   part def Hypervisor {
7.     attribute major_frame : Integer;
8.     part partitions : Partition [*];
9.   }
10.  part def Partition {
11.    port ports : PartitionPort [*];
12.    attribute id : Integer;
13.    attribute name : String;
14.    attribute duration : Integer;
15.    attribute offset : Integer;
16.    attribute period : Integer;
17.    attribute image : String;
18.  }
19.
20.  port def PartitionPort {
21.    attribute msg_size : String;
22.  }
23.  port def QueuingPort :> PartitionPort;
24.  port def SamplingPort :> PartitionPort;
25. }
  
```

Fig. 5: SysML v2 Linux hypervisor library

B. SysML v2 Front-Ends

After creating a SysML v2 library, the corresponding types can be used to model the server-client example. For the demonstration, the proof-of-concept implementation was used to instantiate the hypervisor model as shown in Fig. 6. In the example, a *LinuxHypervisor* is modeled that has partitions for the server and client. Each partition property is redefined (*:>>*) with a specific value. For instance, the period of the partition *ping_server* is set to be 700 ms and therefore fits to the *major_frame* of *LinuxHypervisor*. Each partition also has a request and response port. These ports connect the partitions to each other as displayed in Fig. 6.

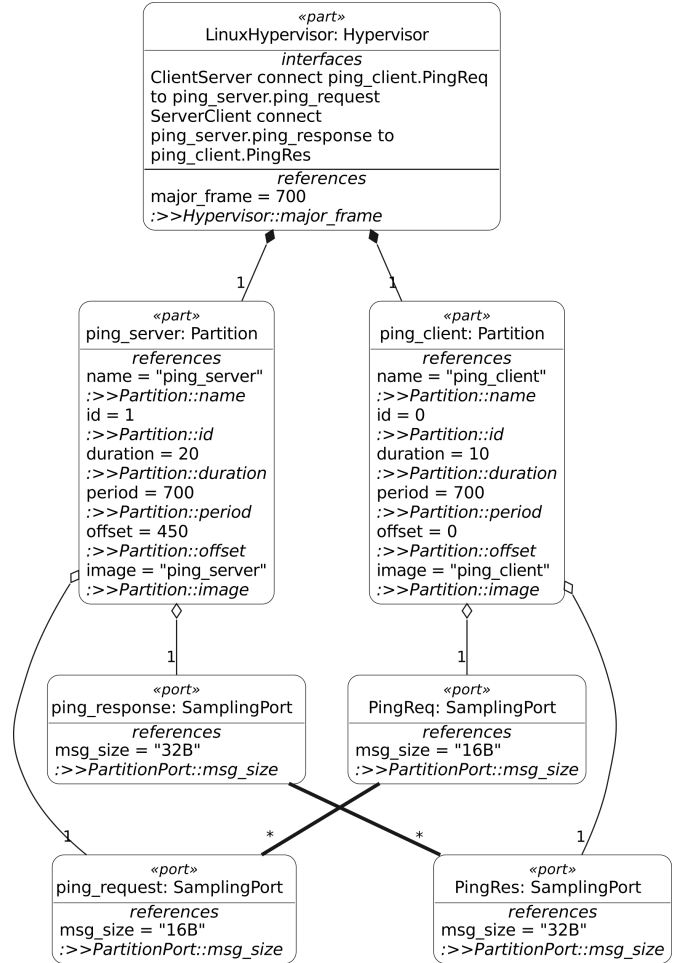


Fig. 6: SysML v2 model of LinuxHypervisor

```

API base path: http://sysml-v2.ft-ssy-stonks.intra.dir.de
Processing..
Posting Commit (268 elements)...550eedc2-859c-4a77-b102-b65df0d0b04a

Saved to Project LinuxHypervisorConfig Fri May 31 11:45:35 UTC 2024 (aebc5d1a-737
4-426b-87d1-9d80d0b8bcf1)
  
```

Fig. 7: Commit of LinuxHypervisor model to server

After all necessary information is added, the model can be uploaded to the server. This functionality is also included in the proof-of-concept implementation and used to publish the model in the form of a commit as shown in Fig. 7. Due to the git-like server behavior, dedicated identifiers are created for the commit and the project.

The model that was uploaded on the server can then also be accessed using arbitrary tools. For instance, we can take a look at the uploaded projects and elements on the server with

²ARINC 653 Linux Hypervisor: <https://github.com/DLR-FT/a653rs-linux>

#	qualifiedName	@id
0	LinuxHypervisorConfig	2e9731da-3cdf-4404-9f66-b268cb6bedd2
1	LinuxHypervisorConfig::LinuxHypervisor	3c4af46d-a719-480e-9ef4-73ef9610a12d
2	LinuxHypervisorConfig::LinuxHypervisor::ClientServer	57e2d1cd-ce2e-4703-9eee-eb7a646c6f37
3	LinuxHypervisorConfig::LinuxHypervisor::ClientServer::consumerPort	baeb787d-a884-4d40-828d-c52d859f9477
4	LinuxHypervisorConfig::LinuxHypervisor::ClientServer::supplierPort	b24df03-28fc-4dfe-b210-423c4df161ba
5	LinuxHypervisorConfig::LinuxHypervisor::ServerClient	30fdb7c3-681d-4232-8dad-dc012c21a5a4
6	LinuxHypervisorConfig::LinuxHypervisor::ServerClient::consumerPort	3d0bd4e4-f4bd-4e93-9b25-b1e27bfc924
7	LinuxHypervisorConfig::LinuxHypervisor::ServerClient::supplierPort	b3db2694-a013-45b9-9213-d2a45389b421
8	LinuxHypervisorConfig::LinuxHypervisor::major_frame	de6c189b-3446-4443-9766-32e3dcb36a4
9	LinuxHypervisorConfig::LinuxHypervisor::ping_client	6e625d3a-7d73-4ecb-85d5-752c13012cee
10	LinuxHypervisorConfig::LinuxHypervisor::ping_client::PingReq	e58cda70-a055-4567-8e60-98616603c9fd
11	LinuxHypervisorConfig::LinuxHypervisor::ping_client::PingReq::msg_size	25e0deca-0963-4e45-a60f-c2ad87c6a71a
12	LinuxHypervisorConfig::LinuxHypervisor::ping_client::PingRes	91c798a3-78f2-4856-8249-86c1cca08ba3
13	LinuxHypervisorConfig::LinuxHypervisor::ping_client::PingRes::msg_size	8d5e7149-a8e4-42bd-99ce-84eae33bb1fa
14	LinuxHypervisorConfig::LinuxHypervisor::ping_client::duration	8a9e6aab-62cc-4823-a802-adfd0a8b5a7e
15	LinuxHypervisorConfig::LinuxHypervisor::ping_client::id	05bac07f-c3f6-4626-8fb4-faf4debbab5c8
16	LinuxHypervisorConfig::LinuxHypervisor::ping_client::image	f7a27ea1-bac2-4dc8-bbe7-7f476dc58c35
17	LinuxHypervisorConfig::LinuxHypervisor::ping_client::name	16da04a6-2feb-471b-9602-1663e933a37d
18	LinuxHypervisorConfig::LinuxHypervisor::ping_client::offset	ebb1f26f-e761-4b35-bedc-9fd8d10e2a8
19	LinuxHypervisorConfig::LinuxHypervisor::ping_client::period	4742775d-583a-4d30-b577-171d48605b00

Fig. 8: SysML v2 server elements

the open-source SysML v2 adaptation of the nu-shell³. The server projects are shown in Fig. 9 and include the previously committed project with the id “aebc5d1a-[...]”. In addition, an extract of the project elements is shown in Fig. 8.

```
95ff53f7-8886-4413c-8a8c-3b69adb777d9 LinuxHypervisorConfig Tue Apr 30 15:07:22 UTC 2024
aebc5d1a-7374-426b-87d1-9d80d0b8b4f1 LinuxHypervisorConfig Fri May 31 11:45:35 UTC 2024
d458635c-32b5-40c2-b159-33e380cef5f2 LinuxHypervisorConfig Mon May 06 08:56:00 UTC 2024
```

Fig. 9: SysML v2 server projects

C. SysML v2 Server Backend

As demonstrated in the previous section, the SysML v2 models are uploaded and managed by a server backend. With the nu-shell SysML v2 front-end, some of the server properties are also shown. Each component of the SysML v2 model results in an identifiable server element as shown in Fig. 8. This confirms the server notation view of Fig. 1c. At the same time, the server stores multiple projects in a git-like format and with distinct identifiers as shown in Fig. 9.

The key component for creating, updating, or deleting information is the standardized API. By using a commit process, the integration with CI/CD concepts is enabled. Because, the goal of this example is to generate an executable configuration for the ARINC 653 Linux hypervisor, a CI/CD pipeline similar to Fig. 4 can be created.

D. Avionics Domain Tools

For the configuration process, a domain-specific implementation was used. The corresponding pseudo-code is shown in Alg. 1. This implementation uses the model information as input and converts it into a configuration that is compliant with the ARINC 653 Linux hypervisor. To get the information from the SysML v2 server, the HTTP/REST API is utilized. As a result, an executable configuration is generated as shown in Fig. 10. In this example, a programming language was used to connect to the model server. However, domain-specific tools can also be connected as long as an interface to the model server is implemented.

³SysML v2 nu-shell: <https://github.com/DLR-FT/sysml-v2-nu>

Algorithm 1: Data access and config. generation

```
Input: project_id
Output: yaml_config
// SysML v2 server http request
 $\mathcal{E} = \text{getElements}(\text{project\_id})$ 
// Sort elements
forall  $e$  in  $\mathcal{E}$  do
  if  $e$  isinstance(partition) then
    |  $\mathcal{P} = \mathcal{P} \cup e$ 
  else if  $e$  isinstance(attribute) then
    |  $\mathcal{A} = \mathcal{A} \cup e$ 
  else if  $e$  isinstance(interface) then
    |  $\mathcal{I} = \mathcal{I} \cup e$ 
// Generate yaml configuration
yaml_config = generateConfig( $\mathcal{P}, \mathcal{A}, \mathcal{I}$ )
```

VI. OPPORTUNITIES AND LIMITATIONS

Considering the characteristics of SysMLv2, the language comes with a lot of opportunities. One example is the improved information exchange between various model-based tool environments because of the standardized SysML v2 API. The improved interoperability also facilitates the tool development for open-source communities, universities, and small companies. This ability could also greatly benefit the available tool landscape. For instance, specific avionics development environments can be built using the SysML v2. At the same time, domain-specific libraries for SysML v2 can be created and published on open-source platforms. Finally, the language comes with enhanced capabilities and will likely be stable, maintained, and widely applied like its predecessor.

Even though the SysML v2 is promising, there are still a lot of areas that will need to improve over time. Due to the novelty of the language, the industry and open-source tool support is just getting started. Potentially challenging aspects for these tools are the following. To make the interoperability a reality, the data needs to be maintained in the same state at every tool. If this works over a server, data needs to be

```

1. major_frame: 700ms
2. partitions:
3.   - id: 0
4.     name: ping_client
5.     duration: 10ms
6.     offset: 0ms
7.     period: 700ms
8.     image: ping_client
9.
10.  - id: 1
11.    name: ping_server
12.    duration: 20ms
13.    offset: 450ms
14.    period: 700ms
15.    image: ping_server
16. channel:
17.   - !Sampling
18.     msg_size: 16B
19.     source:
20.       partition: ping_client
21.       port: PingReq
22.     destination:
23.       - partition: ping_server
24.         port: ping_request
25.   - !Sampling
26.     msg_size: 32B
27.     source:
28.       partition: ping_server
29.       port: ping_response
30.     destination:
31.       - partition: ping_client
32.         port: PingRes

```

Fig. 10: Generated hypervisor configuration

committed frequently and clearly identifiable for each tool. When server commits and requests are happening frequently, the performance of the server needs to meet the corresponding requirements. Similarly, the API and query language need to be expressive and allow a precise selection of model elements.

In addition to the general discussion topics, the safety-critical avionics perspective adds another layer of complexity. One important topic is the possibility for tool qualification according to DO-330 [6]. In this regard, using a central model server might add multiple challenges because the qualification of such a server is unlikely. If the server data is used to create critical artifacts, a workaround is needed. One way could be to request and store the necessary server data locally. The local data could be inspected before executing a lightweight and qualified tool. Another aspect that needs to be considered for a central server are distributed developments with multiple companies. Here, a clearly defined workflow, data storing, versioning, and configuration management are necessary.

Another interesting concept for qualification could be the management of the textual notation of SysML v2 in a git-environment. Reason being that the management of textual artifacts is a scalable and well understood concept in the software community. The ability to read and visualize the textual notation could also prevent tool qualification problems. Unfortunately, the current version of the textual notation only incorporates a subset of the server data as visible in Fig. 1 and misses essential information such as the unique identifiers. Moreover, the standardized exchange is defined over the API and not the textual notation.

To bridge the gap between the general-purpose language of

SysML v2 and the domain-specific tools, interfaces need to be defined and implemented. One way to bridge this gap could be the definition of a domain-specific library by borrowing concepts from DSLs like OAAM. The integration of SysML v2 with DSLs should be further evaluated in future work.

VII. CONCLUSION

The presented paper explores the application of SysML v2 in avionics engineering to address fragmented developments and enhance model-based engineering practices. By connecting domain-specific tools through a standardized API, the concept allows for flexible integration in the current engineering landscape while maintaining the unique advantages of each tool. The concept was demonstrated with the configuration of an open-source ARINC 653 Linux hypervisor, showcasing the practical potential of SysML v2 for avionics developments.

While the development of SysML v2 is still ongoing, the findings suggest that the modeling language offers a standardized and interoperable solution for avionics engineering. It facilitates collaboration between tools and encourages innovation in the development of safety-critical avionics systems. However, some domain-specific aspects such as tool qualification, distributed development, and DSL integration have to be refined in future work.

REFERENCES

- [1] D. Cofer, "Taming the complexity beast," *ITEA Journal*, volume 36, pages 313–318, 2015.
- [2] T. Gaska, C. Watkin, and Y. Chen, "Integrated Modular Avionics - past, present, and future," *IEEE Aerospace and Electronic Systems Magazine*, volume 30, number 9, pages 12–23, 2015. DOI: 10.1109/MAES.2015.150014.
- [3] D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortwell, *SYSTEMS ENGINEERING HANDBOOK, A GUIDE FOR SYSTEM LIFE CYCLE PROCESSES AND ACTIVITIES*. WILEY, 2015.
- [4] B. Annighoefer, M. Halle, A. Schweiger, *et al.*, "Challenges and ways forward for avionics platforms and their development in 2019," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, 2019, pages 1–10. DOI: 10.1109/DASC43569.2019.9081794.
- [5] RTCA, *Integrated Modular Avionics (IMA) development guidance and certification considerations*, RTCA/DO-297, Standard, 2005.
- [6] RTCA, *Software tool qualification considerations*, RTCA/DO-330, Standard, 2011.
- [7] ARINC, *Avionics application software standard interface, part 0, overview of ARINC 653*, Standard, 2021.
- [8] B. Lukić, A. Ahlbrecht, S. Friedrich, and U. Durak, "State-of-the-art technologies for integrated modular avionics and the way ahead," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pages 1–10. DOI: 10.1109/DASC58513.2023.10311229.

- [9] S. H. VanderLeest, "Arinc 653 hypervisor," in *29th Digital Avionics Systems Conference*, 2010, 5.E.2-1-5.E.2-20. DOI: 10.1109/DASC.2010.5655298.
- [10] B. Lukić, S. Friedrich, T. Schubert, and U. Durak, "Automated configuration of arinc 653-compliant avionics architectures," in *AIAA SCITECH 2024 Forum*. DOI: 10.2514/6.2024-1856.
- [11] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems modeling language (SysML v2) support for digital engineering," *INSIGHT*, volume 25, number 1, pages 19–24, 2022. DOI: <https://doi.org/10.1002/inst.12367>.
- [12] Object Management Group, *Systems Modeling Language (SysML©) v2 Request For Proposal (RFP)*, 2017.
- [13] S. Friedenthal, "Requirements for the next generation systems modeling language (SysML©v2)," *INSIGHT*, volume 21, number 1, pages 21–25, 2018. DOI: <https://doi.org/10.1002/inst.12186>.
- [14] Object Management Group, *Systems Modeling Language (SysML©) v2 API and Services Request For Proposal (RFP)*, 2018.
- [15] T. Weilkiens and C. Muggeo, *THE ABSOLUTE BEGINNER'S GUIDE to SysML v2*. INCOSE UK, 2023.
- [16] B. Annighoefer, *An open source domain-specific avionics system architecture model for the design phase and self-organizing avionics*, 2019. DOI: <https://doi.org/10.4271/2019-01-1383>.
- [17] B. Annighoefer and M. Brunner, "Open source domain-specific model interface and tool frameworks for a digital avionics systems development process," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, pages 1–10. DOI: 10.1109/DASC52595.2021.9594380.
- [18] B. Annighoefer, M. Brunner, J. Schoepf, B. Luettig, M. Merckling, and P. Mueller, "Holistic IMA platform configuration using web-technologies and a domain-specific model query language," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pages 1–10. DOI: 10.1109/DASC50938.2020.9256726.
- [19] V. Tietz, J. Schoepf, A. Waldvogel, and B. Annighoefer, "A concept for a qualifiable (meta)-modeling framework deployable in systems and tools of safety-critical and cyber-physical environments," in *2021 ACM 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021, pages 163–169. DOI: 10.1109/MODELS50736.2021.00025.
- [20] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, and F. Kordon, "Validate, simulate, and implement ARINC653 systems using the AADL," in *Proceedings of the ACM SIGAda Annual International Conference on Ada and Related Technologies*, series SIGAda '09, Saint Petersburg, Florida, USA: Association for Computing Machinery, 2009, pages 31–44. DOI: 10.1145/1647420.1647435.
- [21] K. Litwin, I. Amundson, D. Verma, and T. McDermott, *Transforming AADL models into SysML 2.0: Insights and recommendations*, 2024. DOI: <https://doi.org/10.4271/2024-01-1947>.
- [22] J. Hugues, "AADLv2 library for SysMLv2," Carnegie Mellon University, Pittsburgh, PA, USA, Technical Report CMU/SEI-2023-TN-001, Apr. 2023.
- [23] M. Halle and F. Thielecke, "Model-based transition of IMA architecture into configuration data," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pages 1–10. DOI: 10.1109/DASC.2016.7777950.
- [24] F. Schade, T. Dörr, A. Ahlbrecht, V. Janson, U. Durak, and J. Becker, "Automatic deployment of embedded real-time software systems to hypervisor-managed platforms," in *2023 26th Euromicro Conference on Digital System Design (DSD)*, 2023, pages 436–443. DOI: 10.1109/DSD60849.2023.00067.
- [25] M. Halle and F. Thielecke, "Bus network architecture and technology optimisation for avionic systems," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pages 1–8. DOI: 10.1109/DASC50938.2020.9256482.
- [26] C. B. Watkins, J. Varghese, M. Knight, J. Ross, J. Kahn, and B. Petteys, "Data-message modeling for multi-lane architectures on an ima platform using the eSAM method," in *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, 2022, pages 1–10. DOI: 10.1109/DASC55683.2022.9925816.
- [27] T. Dörr, F. Schade, A. Ahlbrecht, *et al.*, "A behavior specification and simulation methodology for embedded real-time software," in *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2022, pages 151–159. DOI: 10.1109/DS-RT55542.2022.9932069.
- [28] Y. Uludağ, Ö. Bayoğlu, B. Candan, and H. Yılmaz, "Model-based IMA platform development and certification ecosystem," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pages 1–11. DOI: 10.1109/DASC58513.2023.10311115.
- [29] M. Halle and F. Thielecke, "Avionics next-gen engineering tools (AvioNET): Experiences with highly automatised and digital processes for avionics platform development," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, pages 1–8. DOI: 10.1109/DASC52595.2021.9594509.
- [30] K. Abdo, J. Broehan, and F. Thielecke, "A model-based approach for early and continuous validation of avionics platforms up to virtual products and hybrid platforms," in *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, 2022, pages 1–10. DOI: 10.1109/DASC55683.2022.9925797.
- [31] C. B. Watkins, J. Varghese, M. Knight, J. Ross, J. Kahn, and B. Petteys, "Auto-derivation of functional flow block diagrams from system architecture using the eSAM method," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pages 1–10. DOI: 10.1109/DASC58513.2023.10311180.