Towards a Model-Based Systems Engineering Framework for the Design and Configuration of Communication Networks in a Data-Driven and Interconnected Aircraft Cabin

Fabian Giertzsch
fabian.maximilian.giertzsch@tuhh.de
Institute of Aircraft Cabin Systems
Hamburg University of Technology
Hamburg, Germany

Marvin Blecken
marvin.blecken@tuhh.de
Institute of Aircraft Cabin Systems
Hamburg University of Technology
Hamburg, Germany

Ralf God
ralf.god@tuhh.de
Institute of Aircraft Cabin Systems
Hamburg University of Technology
Hamburg, Germany

ABSTRACT

In modern aircraft cabin designs, data-driven services, such as predictive maintenance or optimized catering, are being introduced to improve operational and business processes on board aircraft. Along with a communication protocol for such services, the novel Cabin Secure Media Independent Messaging (CSMIM) aviation standard specifies a data model to organize and address available sources and sinks of information in the aircraft cabin. This paper presents an approach for integrating CSMIM-specific concepts in a System Modeling Language (SysML)-based aviation model-based systems engineering (MBSE) framework. For this, a profile-oriented approach is used that extends the SysML and forms the Cabin Data and Communication Modeling Language (CDACML). CDACML can be used to model the information that is communicated according to the CSMIM data model in a specific, airline-customized network architecture. The novel CDACML is developed in a way that it captures information that can also be used for design analyses such as network load or consistency checks with respect to available information as well as enables automated configuration of CSMIM-compliant network nodes. To guide engineers during system design, validation rules are defined using the Object Constraint Language (OCL), thereby ensuring conformance with the CSMIM standard. An application of the MBSE framework with an illustrative example is used to demonstrate its usability for system engineers.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; • **Networks** → *Cyber-physical networks*; *Application layer protocols.*

KEYWORDS

aircraft cabin networks, network modeling, network configuration, SysML, SysML profile

ACM Reference Format:
Fabian Giertzsch, Marvin Blecken, and Ralf God. 2024. Towards a Model-Based Systems Engineering Framework for the Design and Configuration of Communication Networks in a Data-Driven and Interconnected Aircraft Cabin. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3652620.3687425

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687425

1 INTRODUCTION

The aircraft cabin is subject to increasing development demands because of its significant impact on the airline's business and ecological footprint. Data-driven services are introduced to optimize existing operational and business processes or offer significantly new ones. Examples of such novel passenger, crew and airline data-driven services include predictive maintenance based on estimates of the Remaining Useful Life (RUL) of system components [8] and personalized, consumption-optimized meal ordering from the seat in order to enhance the passenger experience and reduce catering waste. To provide the data-driven service, an aircraft cabin network interconnects components of different systems within the aircraft cabin and beyond. For a predictive maintenance service, system components, such as seat actuators of the seat system, can then send data for collection and evaluation to Maintenance, Repair and Overhaul (MRO) organizations. The communication network with its corresponding nodes for communication management and administration is an essential component of this multi-system architecture. Multi-system in this context refers to systems that are tightly coupled during design and operational phase, e.g., an aircraft, compared to a system of systems that allows for more independence of the systems. When designing such a network in a multi-system environment, it is important to consider the heterogeneous device and supplier landscape as well as variations in the network architecture resulting from airline customization or cabin upgrades. To this end, in the ARINC 853 Cabin Secure Media Independent Messaging (CSMIM) working group, research institutions and standardization bodies are working on common aircraft data models and communication protocols enabling the exchange of information for the aforementioned services [2].

For this paper it is assumed that software library-implementations exists or are under development within those companies that plan to supply CSMIM-compliant system components. The software library handles the interaction based on the specified communication protocol, so is establishing a technical mean to exchange information, but is configurable with respect to the data model. The data model is a kind of Interface Control Document (ICD), describing in particular, which information is exchanged. In this article, a model-based systems engineering (MBSE) approach based

on the System Modeling Language (SysML) is developed to gather the information encoded in the data model, i.e., a collection of ICDs. SysML, as a language commonly used in aviation [10, 11], has been chosen to enable integrating the CSMIM data model directly into aircraft system models. Specifically, the framework allows to describe specific CSMIM network configurations originating from an airline customization process or retro-fit upgrade of an aircraft cabin. Certification requirements in aviation stipulate that intended function of an application must be ensured. Therefore, at minimum, it needs to be known in advance that input information required for a certain application is available on the network and network resources are not exceeded. Corresponding analysis and verification activities can be based on models created by means of the modeling framework presented in this paper. In addition, the model information can also be used for an automated configuration of CSMIM components, thus, reducing, for example, transcription errors.

Details on the specific properties and the development process of networks serving as a communication framework for data-driven services in an aircraft cabin are described in Section 2. Also, because CSMIM is still under development, to the best of our knowledge, no SysML-based modeling approaches are publicly known. Therefore, similar modeling techniques as well as related work on ICD modeling are discussed in Section 3. Based on these insights, Section 4 introduces the concept for an MBSE approach build upon a combination of SysML profiles and libraries to capture the specifics of CSMIM-based networks is introduced. In order to show the practicability of the chosen approach, the results of an application of the developed framework on an illustrative example are discussed in Section 5. A conclusion including directions for further research is presented in Section 6.

2 STANDARDIZED INFORMATION-CENTRIC AIRCRAFT CABIN NETWORKS

Current trends in aircraft cabin innovation pursue the optimization of business processes by developing data-driven services. This type of service is realized by a set of cooperating information sources and sinks [15] forming an information-centric aircraft cabin network. Sources provide certain information that is made available through a network and can be consumed by a sink that further processes the information. A sink can, in turn, then also act as source of information by injecting the processing result into the network. Thus, data-driven services do not specifically require certain communication links to be established, however, these services specify the need to access certain information [14]. For example, components installed in the aircraft cabin can provide usage information or electrical parameters such as power consumption allowing a predictive maintenance application to compute an estimate on the RUL [8]. This example already touches upon the challenges raised when designing and configuring a shared information-centric network in an aircraft cabin.

Software that processes source information can be distributed in the aircraft cabin or centrally hosted, e.g., on a multi-purpose cabin server. Information processing can even be further centralized, by temporarily storing corresponding data on the aircraft for later being uploaded to a ground-based, fleet-wide MRO service. Furthermore, the architecture might evolve over the lifetime of the aircraft, e.g., by means of a retro-fit installation of the aforementioned cabin server for hosting applications provided by different suppliers. In case of the predictive maintenance example, a RUL prediction application can be implemented by the component supplier or a third-party, e.g., an MRO organization. In addition, various parts installed in an aircraft cabin, such as passenger seats, are available from different manufacturers while providing the same or similar function. Passenger seats might be directly integrated into the shared network or hidden behind a translator-proxy for legacy, propriety protocols. That is, in case of an information-centric approach a manufacturer-agnostic application can be implemented interfacing with any passenger seat, e.g., showing whether it is occupied, without being configured for the specific network architecture.

As further outlined in the following subsection, the ARINC 853 CSMIM communication standard [2] that is expected to be completed in 2025 [1], can be used as a building block for an information-centric communication network. It provides a data model for describing information provided by sources as well as a communication protocol in order to interact with the data model in a standardized way. Due to its design, CSMIM provides means to connect source and sinks during run-time. However, due to aviation certification requirements, for certain functions, it has to be known in advance if that specific function is actually available in a specific aircraft cabin configuration and if the offered traffic exceeds the physical capabilities of the network. That is, it needs to be verified, if the sources of information required as an input for the function are available in the aircraft cabin configuration and that the resulting data exchange does not overload the network.

2.1 CSMIM Concept

The CSMIM specification is built around an object-oriented data model. For a better understanding of the CSMIM concept itself, in Figure 1, the data model is shown in informal notation. Here, boxes refer to UML class-like concepts with corresponding composition relations. A dash-point line type depicts a relation that exists between classes for which the related assertion on property values holds true. Each interaction on the network is linked to a uniquely identifiable object. A CSMIM server exposes an object that can be accessed by a CSMIM client through CSMIM operations. CSMIM client and server are hosted on CSMIM nodes connected to an IP-based network.

CSMIM operations are specified following the request-response pattern, independently of a specific application layer protocol as per Open Systems Interconnection (OSI) model [19]. Due to this layered approach basically any application layer protocol could implement the CSMIM operations, including those not following the request-response pattern. The current CSMIM specification, however, only defines a mapping onto the MQTT version 5 protocol [9]. Finally, in order to create a well-defined interface, the specification introduces a Concise Binary Object Representation (CBOR)-based serialization of data model elements [4] to be exchanged via MQTT.

In order to realize the concept of information-centric networks, CSMIM introduces the notion of object type specifications as part of the specification view depicted in Figure 1. Objects exposed by CSMIM servers are instantiated from object types. Thus, the

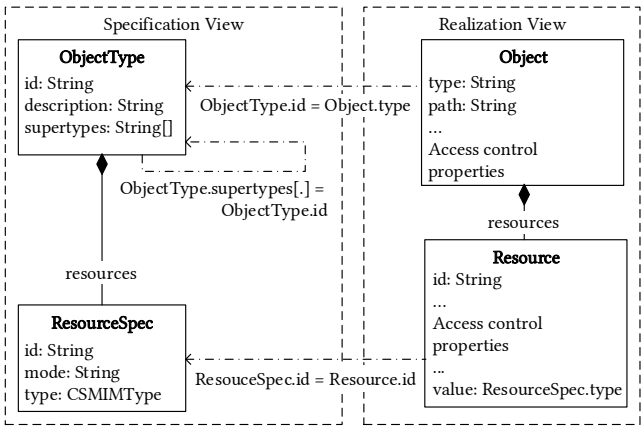


Figure 1: Informal overview of the CSMIM data model, adapted from [2].

specification view forms a structured ICD of a CSMIM server. Object types are uniquely identified by its *id*[1]. By setting its *type* property to the *id* of an object type an object declares conformance to the referenced object type specification. As multiple objects, e.g., each passenger seat, can implement the same passenger seat object type, the *path* property allows addressing a specific object. The CSMIM specification defines means to ensure uniqueness of *path*s during run-time of the network. One core property of the object type model is subclassing which allows base types to be standardized for maximized interoperability. In Figure 1 this is depicted as the *supertypes* reference between the object type concept with itself. In the passenger seat example, a base seat object type can characterize the interface every passenger seat has, e.g., seat occupation. Thus, a seat manufacturer-agnostic fleet-wide seat occupation application can be implemented against that object type, showing, in a multi-class cabin layout, for each seat, if it is currently occupied.

As shown by the *resources* reference in Figure 1, objects are composed of resources which form the atomic building block of the data model. Resources have a *value* that can be accessed through CSMIM operations. The value is its core property that represents the run-time state of the resource. Its *type* and access *mode*, i.e., if the resource is readable, writable or executable, is defined in the resource specification. The types of the value are limited to well-known primitive types such as integer or string as well as the basic composing types array and dictionary. Readable resources can, for example, be sensors that feed a RUL predictor [8], that is, the value of the corresponding resource holds the sensor reading and is accessible through CSMIM operations. Writable resources are similar, but cannot only be updated internally on the server, but its value can also be set through CSMIM operations. Executable resources represent a kind of remote function call, thus, a corresponding CSMIM operation allows to carry function arguments, the value of the resource, in turn, represents the return value of the function execution.

[1]Throughout this paper, when referring to classifiers and attributes either defined in UML/SysML or informal diagrams, the names are typeset in *italics*. In contrast, whenever referring to their abstract counterpart, the name is typeset in roman.

As for the *id* of the object type, the *id* of the resource specification uniquely identifies a resource within the namespace of an object type. The resource, in turn, uses its *id* property to reference its specification.

The current CSMIM specification explicitly notes that the properties of objects and resources are abstract characteristics describing objects and resources realized during run-time of the CSMIM network. The actual value of these data model properties is not required to be stored on some CSMIM node. Furthermore, object type specifications themselves are only intended for implementation-time and are only referenced through their identifier by objects and resources during run-time to show conformance. The latter allows a CSMIM client to identify those objects on the network compatible with the functions provided by it. For example, the seat occupation application, as introduced above, can specifically request from all CSMIM servers that expose a base seat object type whether a seat is occupied.

The data model, in its realization view, also defines properties for managing access to objects and resources during run-time of a CSMIM network. Further properties are discussed when defining the Cabin Data and Communication Modeling Language (CDACML) in Section 4. It should also be noted that only the characterizing properties of the CSMIM data model are considered in this paper. For details refer to [2].

2.2 Research Questions

Considering the description of the CSMIM concept in the previous section, this paper is mainly motivated by the following two research questions:

- **RQ1:** *How can the CSMIM concept be integrated into a SysML-based system development?*
  Developing CSMIM-based data-driven services in SysML-based multi-system architectures require joint development between system engineers and network design engineers. To facilitate communication and enabling model-based verification and analysis, it is, thus, desirable, to incorporate CSMIM into a SysML-based system development.
- **RQ2:** *How can the modeler be guided to follow the rules defined by the CSMIM-Standard?*
  In order to ease the integration effort of multi-system architectures, the impact of design flaws resulting from misinterpretations and incorrect application of the CSMIM standard can be reduced by guiding the modeler. To this end, means to validate the rules as specified in CSMIM standard are required.

The answers to these research questions provided in this paper also summarize the contributions of this paper.

3 FUNDAMENTALS AND RELATED WORK

The CSMIM specification combined with concrete data type specifications and instantiated objects forms a well-defined ICD from OSI layer four [19], i.e., transport layer, and upwards. This includes a transformation of abstract information into data, according to the Data, Information, Knowledge, Wisdom (DIKW) hierarchy [14, 29], through the data model and its CBOR encoding. As CSMIM is media-independent no information is known on deployed data link and

physical layer protocols as well as IP addressing. By means of CDACML the concrete object types and objects shall be formalized to support creating coherent ICDs and tracing them towards the corresponding information flow in the the system model for early verification as well as configuration.

In [32], approaches on how ICDs can be specified have been studied based on an extensive literature review and expert interviews. These approaches are not limited to actual document-based ICDs, but also model-based methods are considered. It is concluded that the most suitable approach for capturing ICDs depends on the particular use case. According to the definition in [32], this paper can be described best by a decoupling engineering activities use case, with focus on the feature of reducing of ambiguities. In the context of CSMIM networks, decoupling engineering activities refers to different companies concurrently designing components intended to exchange information based on exposed CSMIM objects. As it is expected that manufacturers of such components implement software-libraries to handle CSMIM operations, this paper focuses on the coordination of the object types and objects available in a specific aircraft cabin customization. As a result of applying the method proposed in [32], this paper focuses on developing CDACML to guide engineers to follow the rules imposed by CSMIM with the goal of generating a consistent, formal and semantic ICD.

To certify a cooperating and collaborating multi-system for installation in an aircraft, authorities require traceability and consistency between design artifacts [30, 31]. This includes tracking the configuration of components in each aircraft cabin version, customized for an airline. The novel CDACML developed in this paper allows capturing a specific CSMIM cabin configuration defined by the CSMIM objects. As shown in [33], system models can generally be used to generate configuration files for the software of the components described by the system model. Furthermore, as outlined in [18], models can even be used to automatically provision the modeled components and monitor, if the deployed system conforms to the model.

In the following section CDACML is introduced as a language extension of SysML. As discussed in [5], if particular modeling needs arise either a Domain-Specific Language (DSL) can be developed or a General-Purpose Language (GPL), e.g., Unified Modeling Language (UML) [5, 26] or SysML [27], can be extended. According to [5], an advantage of a DSL is that modelers do not need to understand the full range of modeling concepts a GPL can provide. However, in aviation industry the SysML, which is a GPL, is commonly used to create system models [10, 11]. In order to support generation of configuration for system components under development, CDACML is specifically designed to be closely linkable to that system model, see also (**RQ1**). Thus, modelers, more specifically system engineers, can use CDACML directly in their modeling tool and no model-to-model transformations are necessary.

Research related to SysML modeling of networked systems ranges from approaches focusing on developing the multi-system itself to detailed specification of communication stacks. In [7] a SysML profile for modeling service-oriented system-of-systems is presented. CSMIM object types could be considered as a kind of service description. The described interface descriptions, however, are directly linked to protocols and encoding, but do not originate from a common data model as for CSMIM. In [12] an MBSE method for satellite communication systems including consideration of communication protocol stacks is provided. Similarly to [7], no traceability towards a data model that defines a transformation from information to data is inherently supported. Others also develop MBSE methods for specific multi-system environments, e.g., [21], however, focus on the integration of existing, heterogeneous interfaces.

Moreover, approaches for extracting ICDs of generic SysML interfaces, i.e., not being protocol specific, described by the behavior of the realizing classifier exist [17]. However, CDACML is not intended to capture any type of a software interface, but is specifically designed for CSMIM interfaces and its goal is to guide the modeler through the system design process to ensure a consistent CSMIM network architecture, see (**RQ2**). The importance of consistency in aviation is also discussed in [16]. However, the developed SysML profile is limited to network configuration up to OSI layer 3, i.e., network layer. The research in [16] is based on [34] that provides SysML viewpoints for system interfaces following the OSI model. The concept of this paper complements the approach in [34] by providing means to model how the abstract information is encoded in case of a CSMIM interface.

As introduced in the previous section, CDACML needs to capture object types such that they can be distributed as a library for the use by system engineers. Furthermore, concrete objects that comply to the specification of an object type need to be modeled. CDACML is inspired by the Risk Analysis and Assessment Modeling Language (RAAML) that is specified as an extension of SysML [28] and uses a combination of a minimal SysML profile and a SysML library. This concept allows to take advantage of structural modeling capabilities of existing SysML tools [28] such as composing or specializing blocks. However, to better guide the modeler, a profile-oriented approach, similar to [3], is taken in this paper. This allows to add tag definitions[2] to stereotypes and the tagged values can be checked for compliance to requirements imposed by the CSMIM specification using Object Constraint Language (OCL) [6, 24]. Furthermore, by relying on tagged values for describing properties of object types and objects on model level, these properties can be clearly separated from values that only exist during run-time, i.e., on instance level, namely the value of a resource [25]. Validity of this approach is strengthened as the UML specification [26] *Clock* example for stereotypes with tags introduced in Section 12.3.5 shares characteristics with CSMIM modeling requirements. In the example a *Clock* stereotype as an extension of *Class* is created and tags such as *OSVersion* are attached. An instance of a clock, i.e., it exists on model level, is then further described by a specific value assigned to the *OSVersion* tag. The same concept is required to describe object types and objects by specific values, e.g., their identifier, on model level.

[2]To better distinguish between properties of stereotypes and properties of instances of an extended metaclass, e.g., a SysML *Block*, the traditional terminology according to [26] referring to tag definitions and tagged values is used.

4 CABIN DATA AND COMMUNICATION MODELING LANGUAGE

To integrate the CSMIM concepts described in Section 2.1 into a SysML-based system development (**RQ1**), the GPL SysML is extended using extension mechanisms of SysML inherited from UML [26]. As explained in the previous section, a profile-oriented approach is chosen for extending the modeling language to include the CSMIM concepts. Due to the explicit separation of a specification view and a realization view in the CSMIM data model, CDACML has different use cases. To identify these use cases with respect to the views and to describe their application in the model-based system development context, first CDACML modeler roles are specified in Section 4.1. This also includes tracing domain-specific information modeled with CDACML to the system model.

Consequently, CDACML defines profiles for both specification view and realization view, complemented by libraries for both views, introduced in Section 4.2 and Section 4.3, respectively. In addition, for both views OCL is used to define constraints that direct the modeler towards (CSMIM-)standard-conform modeling (**RQ2**). For better readability only characteristic OCL constraints are presented that show the underlying principle of how CDACML is designed to allow formulation of such constraints. These OCL constraints can be implemented as tool-specific validation rules to check conformance, e.g., in Cameo Systems Modeler [23], also used for creating the models for this paper, or Enterprise Architect [35].

4.1 Definition of Modeler Roles

For a SysML-based development approach of systems or interconnected multi-systems various modeling patterns or frameworks exist. To define the role of CDACML in a system development, the process and method-agnostic System Architecture Framework (SAF) developed by the German chapter of the International Council on Systems Engineering (INCOSE) is used [13]. For modeling interactions between systems SAF proposes to use SysML internal block diagrams and ports, either through abstract information flows or definition of the complete OSI stack similar to approach proposed by [34]. Each information flow determines which CSMIM objects are needed to realize the underlying communication through CSMIM. This might require the definition of new objects types or the extension of existing ones. In this context the following three engineering roles have been identified:

- The **multi-system engineer** specifies a goal for a set of interacting systems and formulates corresponding top-level requirements for each system. In this role, the multi-system engineer is responsible that the common goal is achieved. The top-level requirements include the assignment of object types to system-external ports that constitute the foundation of the inter-system interaction.
- The **object type engineer** has a clear understanding of the CSMIM data model and uses this knowledge to define, in collaboration with the multi-system engineer, object types following the best practices encouraged through the CSMIM specification.
- The **system engineer** accepts the top-level requirements including the objects types assigned to the system interfaces and designs the system. Taking into account all possible variations of the aircraft cabin configuration that can also influence the multiplicity of system components, the system engineer models concrete objects, always ensuring that the path of any object is unique.

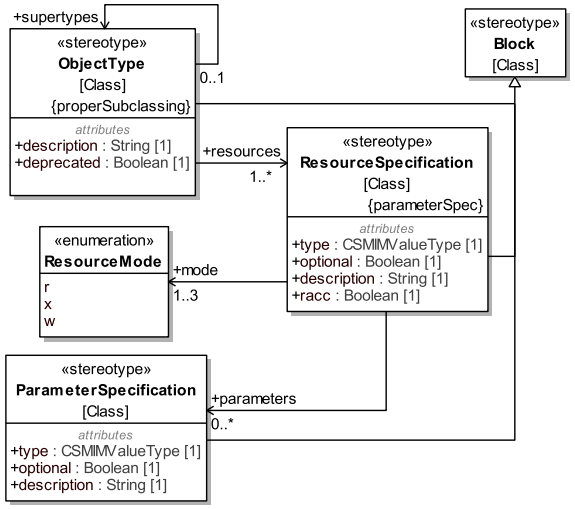


Figure 2: Specification view stereotypes for object type and resource specification as extension of metaclass Class.

4.2 Specification View

According to the CSMIM concept described in Section 2.1, new stereotypes are defined to enable modeling of the specification view. In Figure 2 the extension of the metaclass *Class*[3] for adding the concept of object type specification is depicted. In addition, the stereotype specializes the stereotype *Block* to inherit the properties of a block and to keep CDACML in the language frame of SysML.

The stereotype *ObjectType* denotes the specification of object types. *ObjectType* has the following associated tags: *description*, *deprecated*, *resources* and *supertypes*. These associated tags represent the properties of an object type specification as prescribed by the CSMIM specification [2]. The property *id* introduced in Section 2.1 is not considered as a tag. Identifiers of object types are modeled using the *name* property of the instances of the metaclass *Class* of the elements to which the stereotype *ObjectType* is applied.

The associated tags *description* and *deprecated* do not impose specific requirements to the design of CDACML, CSMIM-specific semantics are described in [2]. The tag *supertypes* is, in contrast, particularly relevant to integrate the concept of subclassing. This tag allows to specify that an object type inherits the properties of a parent object type, e.g., a standardized object type specification. Thus, subclassing allows adding resources to a standardized object type specification to add functionality.

The constraint *properSubclassing* associated with the stereotype *ObjectType* is intended to ensure compliance to the subclassing

[3]To improve clarity of profile diagrams, extension associations are not drawn, but depicted as part of the specialized parent stereotype.

Listing 1: OCL constraint on Objects Types to ensure compliance with CSMIM subclassing rules.

```
context ObjectType inv properSubclassing:
let superResSpec :
    -- Get sequence of all resources, including those inherited
    -- through supertype-chain.
    Sequence(ResourceSpecification) =
    self.supertypes->closure(p|p.supertypes)
        ->select(p|p.oclIsTypeOf(ObjectType))
        .resources->asSequence() in

    -- Iterate through resources of new object type.
    self.resources->iterate(ownedResourceSpecIt; result : Boolean = true |

    -- Get inherited resources that are redefined by ownedResourceSpecIt
    -- of new object type, i.e., name is equal.
    let matchingSuperResourceSpec :
        Sequence(ResourceSpecification) =
        superResSpec->select(resSpec|resSpec.name = ownedResourceSpecIt.name) in
        -- iterate through all redefined resources and check rules.
        if matchingSuperResourceSpec->iterate(matchingSuperResourceSpecIt;
            properSubclassing : Boolean = true |

            -- Resource type must not change.
            if matchingSuperResourceSpecIt.type <> ownedResourceSpecIt.type
            then
                properSubclassing = false
            else
                properSubclassing
            endif
            -- Further checks as required.
        ) = false
        then
            result = false
        else
            result
        endif
    )
```

rules defined by CSMIM. According to [2], subclassing must follow the Liskov principle [20]. In context of CSMIM this refers to applications that rely on resource values of some object type *a* to provide its functionality. The functionality of any such application must not be altered irrespective of whether it uses objects of type *a* or objects of any type sublassing *a* [2]. That is, for example, mandatory resources must not be deleted. The OCL rule shown in Listing 1 shows the pattern on how to check conformance to the Liskov principle for CSMIM object types. For better readability only one rule is checked, namely that the type of a resource must not be changed. If an object type *b* defines another object type *a* as its supertype, a resource that is specified in scope of *a* is considered to be redefined if a resource with the same name, i.e., *id*, is defined in scope of *b*.

An object type specification shall also include a list of resources that the object will have. Objects contain at least one resource, but the number of resources contained in objects is not limited. This is formalized by the multiplicity *1..** which labels the association end of the *resources* tag. To add resources to the corresponding tag, one has to specify resources by applying the *ResourceSpecification* stereotype.

The *ResourceSpecification* stereotype defines the following associated tags as shown in Figure 2: *type*, *optional*, *description*, *racc*, *mode* and *parameters*.

Similar to the identifier of object type specifications there is no tag specifically defined for the identifier of resources. Instead the *name* property of the instances of the metaclass *Class* of the elements to which the stereotype *ResourceSpecification* is applied, is used.

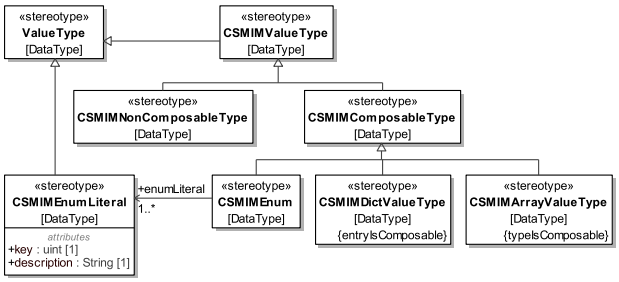


Figure 3: Specification view stereotypes for CSMIM value types as extension of metaclass DataType.

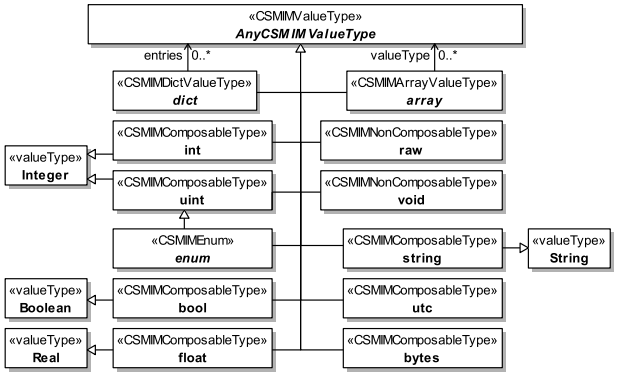


Figure 4: Specification view library containing available CSMIM value types.

The tags *optional* and *description* are self-explanatory, *racc* is required due to technical details of the CSMIM communication protocol [2], but not relevant for the design of CDACML. Apart from these tags, special attention is paid to the tags *type*, *mode*, and *parameters*, as different rules must be followed depending on the specific values assigned to these tags.

Resources have different access modes as described in Section 2.1. The *mode* tag is used to set the access modes for resources. An enumeration denoted as *ResourceMode* defines the different modes, namely readable (*r*), writable (*w*), and executable (*x*). The fact, that the assignment of access modes is not limited to one mode is represented by the multiplicity *1..3* which labels the corresponding association end of this tag.

Each resource also has an associated data type, defined by the identically named tag of type *CSMIMValueType*. The implementation of data types standardized by CSMIM into CDACML is shown in Figure 3 and Figure 4. For executable resources the data type defined in the tag *type* applies to the return value provided by these resources. Possible value types are shown in Figure 4 that is part of the specification library of CDACML. Among well-known primitive types, three types are specifically handled in CDACML, namely, *enum*, *array* and *dict*. For the latter types, corresponding stereotypes, i.e., *CSMIMEnum*, *CSMIMArrayValueType* and *CSMIMDictValueType*, respectively, as shown in Figure 3, are defined.

Listing 2: OCL constraint on CSMIM dictionaries to ensure they are build om composable types.

```
context CSMIMDictValueType inv entryIsComposable:
self->select(not isAbstract)->asSequence()
    ->first().ownedAttribute
    ->forAll(type.oclIsKindOf(
        CSMIMComposableType))
```

Due to the fact, that in CSMIM specific *key*s can be assigned to enumeration literals, UML *Enumeration DataType* could not be used. Thus, a *CSMIMEnumLiteral* stereotype, based on the UML *EnumerationLiteral* concept, is introduced that defines a tag *key* for that purpose. As for the object type and resource identifier, the name of the metaclass *DataType* is used for name of the enumeration literal.

For the *array* and *dict* value type, stereotypes are introduced to place constraints on the *valueType* and *entries* attributes defined in the specification library shown in Figure 4. The *valueType* attribute allows, by redefinition, to specify the type of the array entries and the multiplicity defines its size. Similarly, the *entries* attribute of a *dict* can be subset to construct a dictionary for which the role name of *entries* defines its key.

The value types *enum*, *array* and *dict* as part of the specification library are abstract as the modeler needs to create concrete CSMIM-compliant value types from them.

Due to the serialization of resource values defined by CSMIM, its specification distinguishes between composable types and non-composable types. Only composable types are allowed to construct *array* and *dict*. This requirement can be checked by the OCL constraint *entryIsComposable* shown in Listing 2. A similar constraint *typeIsComposable* can be formulated for the *array* value type.

When accessing executable resources, similar to function calls, parameters can be added to the request. As the value type of executable resources only specifies the return value of that request, a *parameters* tag for zero to unconstrained many *ParameterSpecifcations* is added to *ResourceSpecication*. For this, the *ParameterSpecification* stereotype has the following associated tags: *type*, *optional* and *description*.

The name of parameter is defined by the *name* attribute of the corresponding instance of the metaclass *Class*. Its type is specified by assigning a *CSMIMDataType* to the tag *type*. Additionally, a textual description of the meaning of the parameter value is provided as well as if the parameter is optional.

As shown by two examples, OCL can be used to support the modeler satisfying the requirements established by CSMIM. The CSMIM specification imposes more requirements of that kind, such as allowed combinations of resource access modes or formats for identifiers. However, as these requirements follow similar pattern, CDACML is capable of capturing them.

4.3 Realization View

The realization view uses object types and their corresponding resource specifications to model concrete objects and resources that exist in a CSMIM network. As for the specification view, stereotypes as defined in Figure 5 are used to define the notion of an object and

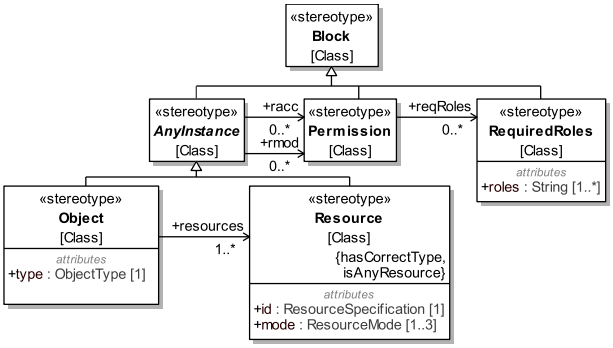


Figure 5: Realization view stereotypes for object and resource as an extension of metaclass Class.

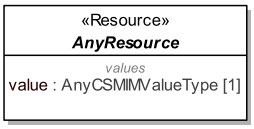


Figure 6: Realization view library containing the parent resource for any concrete resource.

resource. Thus, objects and resources exist on model level and are uniquely identifiable through the path of the object as explained in Section 2.1. As a result, object instances that exist during run-time of a CSMIM network represent singletons with respect to their object.

The *Object* and *Resource* stereotype are both inherited from the *AnyInstance* stereotype that holds the access control tags. CSMIM defines a role-based access control mechanism for reading (*racc*) and modifying (*rmod*) resources. A set of roles required for reading or modifying, respectively, can be formulated through an instance of the *RequiredRoles* stereotype. If different sets of roles are allowed to access a resource, multiple *RequiredRoles* instances can be created and linked through an instance of the *Permission* stereotype. The details on the effect of assigning access rights to an object or a related resource are described in [2].

The *type* and *id* tags of the object and resource, respectively, are used to declare conformance to a specific object type and its resources. CSMIM allows that the modes of a resource of a concrete object can be extended, compared to its specification, e.g., a resource that is specified to be readable, can in addition be writable. This can be modeled through the *mode* tag. Concrete resources can be assigned to its object through the *resources* tag. The unique identifier of an object that is also used as its address denoted as *path* in the CSMIM data model is defined by the *name* attribute of the corresponding instance of the *Object* stereotype.

In contrast to the specification profile defining stereotypes *ObjectType* and *ResourceSpecification* without corresponding instances in a library, an instance of *Resource* stereotype is part of the realization library. As introduced in Section 2.1, a resource has a value attached to it which has a concrete value only during run-time of the CSMIM network stored on a CSMIM server, i.e., whenever an

Listing 3: OCL constraint on Resources to ensure that type of Resource value matches its specification.

```
context Resource

inv isAnyResource:
self->select(not isAbstract)->asSequence()
        ->first().generalization.general
        .name->includes('AnyResource')

inv hasCorrectType:
    self.ownedAttribute->select(name = 'value')
    ->asSequence()->first().type = self.id.type
```

actual instance is created. Thus, as shown in Figure 6, the value property is not defined as a tag of the *Resource* stereotype, but as a SysML *ValueProperty* of an abstract instance of a *Resource* stereotype named *AnyResource*. It is intended that any resource specializes from that *AnyResource Resource* and redefines the *value ValueProperty* with a *ValueProperty* of the type that has been specified for that resource. In order to ensure that the modeler uses CDACML accordingly, two OCL constraints as shown in Listing 3 are added to the *Resource* stereotype. The first invariant *isAnyResource* checks that any resource specializes from *AnyResource*. Knowing that any resource has a *ValueProperty* named *value*, the second invariant *hasCorrectType* validates that the type of the *ValueProperty value* matches the type of the linked resource specification.

5 ILLUSTRATIVE APPLICATION AND DISCUSSION OF CDACML

The application of CDACML is illustrated by a passenger seat connected to a CSMIM compliant cabin network. Various use cases related to the interconnected passenger seat are considered to demonstrate the application of CDACML in all its facets. This includes moving passenger seats into predefined positions for certain operational processes, such as maintenance, or other processes, such as cabin cleaning. For further support of cabin and maintenance crew, an overview of current seat positions can be provided. For safety reasons, during certain flight phases, such as taxiing, takeoff, and landing, passenger seats need to be in a pre-defined position, thus, the remote control of seats must be disabled during these phase of flight. To improve the boarding process and provide cabin crew with an overview of the seat occupancy, the connected seat detects and communicates its occupancy status. Other use cases, such as the collection and communication of sensor data for health monitoring and predictive maintenance of seat actuators are also appropriate for showing the application of CDACML, but do not provide any added value for showing the capabilities of CDACML and are therefore not further considered.

The aforementioned use cases emerged and were modeled during a joint-aviation research project with participants including aircraft manufacturers as well as system and equipment suppliers. These use cases are realized by integrating the seat system in a multi-system architecture which is designed within a MBSE process as outlined in Section 4.1 and not further discussed in this paper. Communication

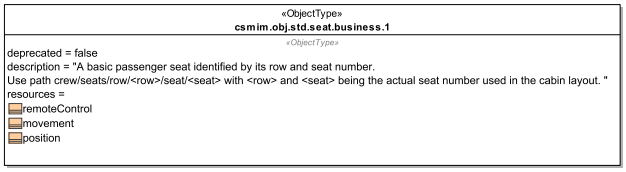
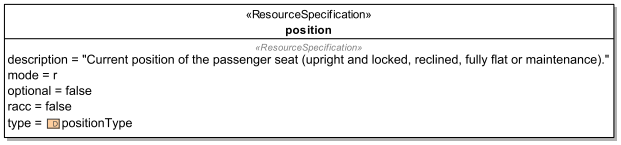
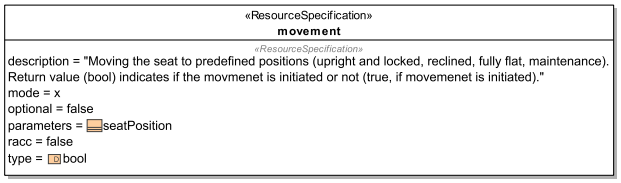


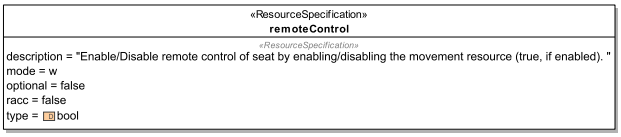
Figure 7: Definition of a standardized Object Type for the seat system.



(a) Specification of readable resource position.



(b) Specification of executable resource movement.



(c) Specification of writable resource remoteControl.

Figure 8: Specification of the resources of the business class seat standard object type csmim.obj.seat.business.1.

in this multi-system architecture is based on the CSMIM protocol, thus, requires the specification of CSMIM objects and resources.

As for the definition of CDACML itself, Cameo Systems Modeler 2024x [22] is used as the SysML modeling tool.

In a first step, a standardized seat object type specification is modeled from an object type engineer's point of view. This standardized seat object type and its associated resource specifications is made publicly available for interoperability and reusability reasons, as described in Section 2.1, and can be subclassed by a seat manufacturer who intends to add specific functionalities required for implementing the use cases. In this example, a standardized object type for a business class seat is modeled as shown in Figure 7.

This standard business class seat object type specification includes three resources, namely *position*, *movement* and, *remoteControl*. These resources are considered essential for the design of business seats and have therefore been added to the standard object type by modeling corresponding resource specifications shown in Figure 8. As defined in the previous section, the names represent the CSMIM data model *id* property.

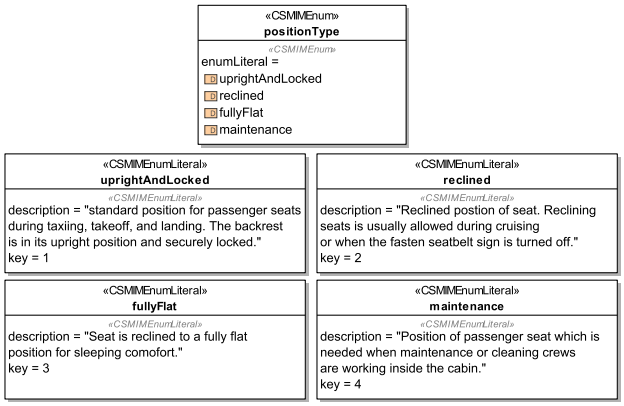


Figure 9: Definition of an enumeration value type capturing possible seat positions.

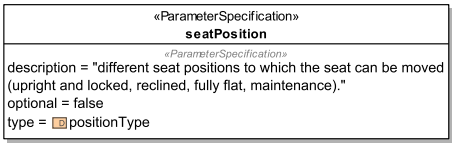


Figure 10: Specification of the function parameter of the executable resource movement.

To provide the cabin crew with an overview of the current seat position, a readable resource named *position* is defined as depicted in Figure 8a. There are different predefined positions any business seat can be in, represented by the current value of the resource. As shown in Figure 9, this is modeled by defining enumeration named *positionType* with enumeration literals representing predefined seat positions. According to the CSMIM specification, in line with the Liskov substitution principle [20], a manufacturer can add literals to this enumeration through subclassing the corresponding object type and redefining the *positionType* enumeration.

In addition to the position resource, an executable resource named *movement* is specified as shown in Figure 8b. For executing this resource, a target seat position into which the seat moves needs to be passed as a parameter. Using CDACML, this is defined by assigning a corresponding *seatPosition ParameterSpecification* to the *parameters* tag as depicted in Figure 10. When executing the resource, *bool* value is returned, as defined by the *type*, to indicate whether the movement has been initiated or not.

Through altering the *remoteControl* resource shown in Figure 8c, the remote control of the passenger seat can be enabled or disabled. Hence, the resource is specified as a writeable resource, i.e., *mode* tag is set to *w*, and contains a *bool* value that can be set.

In case a seat manufacturer integrates additional sensors into the passenger seat, a new manufacturer-specific object type inheriting *csmim.obj.std.seat.business.1* can be defined. In Figure 11 an example of a manufacturer-specific object type adding one readable resource representing if the seat is occupied or not, e.g., provided through an resistor-based occupancy sensor. Through the *supertypes* tag,

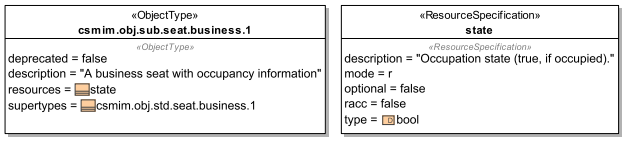


Figure 11: Extending the standard Object Type for a business class seat by adding a manufacturer-specific resource through subclassing.

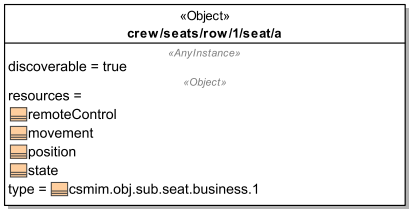


Figure 12: Definition of of an Object that declares conformance to the manufacturer-specific Object Type of a business class seat.

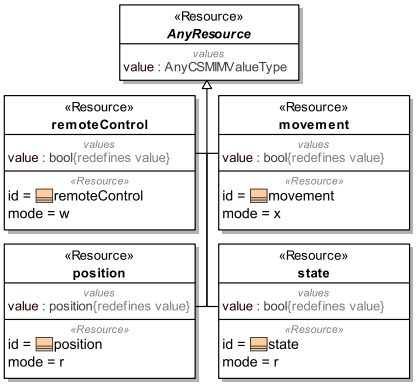


Figure 13: Definition of the Resources that are specified the manufacturer-specific business class seat Object Type.

the object type is linked to the standardized business class seat object type. This also triggers the evaluation of the OCL constraint defined in Listing 1 that checks that all subclassing rules imposed by the CSMIM specification are satisfied.

When all object types are defined or selected from the existing library, respectively, that are needed to be able to satisfy the top-level system requirements related to the system interfaces, cf. Section 4.1, corresponding objects can be generated. For the connected seat use case for each seat installed in a specific cabin configuration, an instance of an *Object* stereotype is created. The name of the object must be unique and defines its path in the CSMIM network. For this object type, its specification prescribes that the path is constructed as *crew/seats/row/<row number>/seat/<seat number>*. One of the resulting objects is shown in Figure 12. As the object declares conformance to the manufacturer-specific object type through its *type* tag, it is composed of four resources assigned to the object

#	Name	Deprecated	Description	Resources	Supertypes	Effective Resources
1	csmim.obj.std.seat.business.1	false	A basic passenger seat identified by its row and seat number. Use path crew/seats/row/<row>/seat/<seat> with <row> and <seat> being the actual seat number used in the cabin layout.	remoteControl movement position		remoteControl position movement
2	csmim.obj.sub.seat.business.1	false	A business seat with occupancy information	state	csmim.obj.std.seat.business.1	remoteControl position movement state

Figure 14: Automatically generated table of modeled object types with a list of all effective resources including inheritance.

through the *resources* tag depicted in Figure 13. Three resources are defined by the supertype, one resource is defined by the object type itself. The OCL constraints specified in Listing 3 ensure that the resources actually specialize *AnyResource* and that the type of the *value* property, i.e., *bool* for *remoteControl*, *movement* and *state* resources and *positionType* for *position* resource, conforms to its specification, cf. Figure 8.

As can be seen from this example use case, the profile-oriented approach limits usage of graphical modeling capability of the SysML modeling tool. No association-like UML/SysML notation is defined for showing the link between two blocks or classes that are related through a tag value, e.g., the link between objects and their resources. However, for a better overview, this kind of model information can also be visualized in tool-specific tables constructed from OCL rules as exemplarily shown in Figure 14. That is, for example, for each object type, including those inherited from supertypes, possibly through a longer inheritance chain, the effective resources, taking re-definition into account, are computed. This is solved by using a tool-independent OCL rule, similar to the one shown in Listing 1 for checking CSMIM subclassing rules. The CDACML presented was, however, not only developed as a means of capturing CSMIM interface information, its intended uses range from configuration management to design support and analysis.

In the context of configuration management, CDACML, for example, enables the exchange of CSMIM object types through SysML libraries to support eliminate inconsistencies between distributed SysML-based design artifacts. Also, through model-to-text transformations, there is no need, for model-external tracking of CSMIM-related information with the same associated risk of inconsistency. As shown for the generation of object type tables, OCL or tool-specific means can be used to extract the required information, e.g., for configuration files that allow to adapt a standard software implementing a CSMIM interface to a specific aircraft cabin configuration, directly from the system model.

With respect to design analyses, CDACML defines the value of a resource as *ValueProperty* on model level, so its value can be set on instance level, e.g., when simulating a specific cabin configuration for validating the multi-system. Furthermore, such simulation or a static model analysis can be used to do an early validation that the resulting network traffic does not overload the network infrastructure. This analysis is not limited to the design of new aircraft cabins, but can also help to evaluate modifications, e.g., updating a seat controller to expose sensor data for RUL prediction or checking that all aircraft of an airline fleet provide objects required as an input for a new application.

6 CONCLUSION AND OUTLOOK

Within this paper a novel SysML profile and library, the Cabin Data and Communication Modeling Language (CDACML), for integrating the CSMIM data model into a SysML-based MBSE framework were presented. CDACML allows to create a library for industry-wide standardized object types. This library can be used by engineers designing a multi-system to specify the ICD of the system interfaces. If required, new object types can be introduced or derived from standardized object types forming a kind of distributed library that can be shared with partners or suppliers on demand. Through accompanying OCL constraints it is ensured that object types are designed in line with CSMIM rules. System engineers can then create concrete objects that state through SysML means conformance to the object type specified by the multi-system engineer. For this, as for the specification of object types, OCL constraints have been developed that support system engineers to comply to the object type specification as well as the generic CSMIM requirements.

In future work it is planned to extend CDACML by more specific means to trace objects towards the system model. If, for example, SysML ports and connectors are used to model connections between system components, objects and their respective resources shall be an integral part of that modeling technique. This allows to extend early validation features of CDACML by further reducing misunderstandings of system ICDs during the actual design activity of a system engineer. Furthermore, also showing the extensibility capabilities of CDACML, in order to reduce the modeling effort, tool-specific plugins can generate objects including all of its, possibly inherited, resources based on a selected object type.

A connected passenger seat use case served as an example to demonstrate the features of CDACML. By means of this illustrative application also limits of the chosen profile-oriented approach, especially with respect to the visualization have been discussed. However, alternative means by creating tables defined through OCL have been proposed. In future work an evaluation of CDACML is required to show to its applicability in an industrial context. For this, the discussed uses of CDACML that also show the value of proposed SysML profile and library, can serve as a baseline for such validation.

ACKNOWLEDGMENTS

This work was supported by the LuFo VI-2 project "i+sCabin2.0: Intelligenter und smarter Kabinenbetrieb" (i_sCabin2, project number 20D2130K) funded by the Federal Ministry for Economic Affairs and Climate Action based on the decision by the German Bundestag.

REFERENCES

[1] Aeronautical Radio Incorporated: Cabin Systems Subcommittee (ARINC CSS). 2021. ARINC Project Initiation/Modification (APIM): Cabin Secure Media Independent Messaging.

[2] Aeronautical Radio Incorporated: Cabin Systems Subcommittee (ARINC CSS). 2024. Draft 2 of ARINC Project Paper 853: Cabin Secure Media-Independent Messaging (CSMIM) Protocol.

[3] Kirsten Berkenkötter and Ulrich Hannemann. 2006. Modeling the railway control domain rigorously with a UML 2.0 profile. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 398–411.

[4] C. Bormann and P. Hoffman. 2020. RFC 8949 - Concise Binary Object Representation (CBOR). https://datatracker.ietf.org/doc/html/rfc8949. Accessed: 2024-03-19.

[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-driven software engineering in practice*. Morgan & Claypool Publishers.

[6] Jordi Cabot and Martin Gogolla. 2012. Object constraint language (OCL): a definitive guide. In *International school on formal methods for the design of computer, communication and software systems*. Springer, 58–90.

[7] Jerker Delsing, Géza Kulcsár, and Øystein Haugen. 2022. SysML modeling of service-oriented system-of-systems. *Innovations in Systems and Software Engineering* (2022), 1–17.

[8] EASA and Collins Aerospace. 2023. *Formal Methods use for Learning Assurance (ForMuLA)*. Technical Report.

[9] Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 2019. MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Accessed: 2024-03-18, Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[10] Marco Ferrogalini. 2019. Modeling and Simulation @Airbus a fundamental digital transformation axis across product, manufacturing and support in service. Presentation at MODELS Conference 2019 – Industry Day 18 September 2019 | Munich - Germany.

[11] Marco Forlingieri. 2022. The Four Dimensions of Variability and Their Impact on MBPLE: How to Approach Variability in the Development of Aircraft Product Lines at Airbus. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* (Florence, Italy) *(VaMoS '22)*. Association for Computing Machinery, New York, NY, USA, Article 15, 4 pages. https://doi.org/10.1145/3510466.3511275

[12] Su Gao, Wei Cao, Luhai Fan, and Jihong Liu. 2019. MBSE for Satellite Communication System Architecting. *IEEE Access* 7 (2019), 164051–164067. https://doi.org/10.1109/ACCESS.2019.2952889

[13] Gesellschaft für Systems Engineering e.V. . [n. d.]. System Architecture Framework (SAF) Specification. https://github.com/GfSE/SAF-Specification. Accessed: 2024-03-18.

[14] Fabian Giertzsch, Christian Hertwig, Uwe Salomon, and Ralf God. 2021. Requirements and Technical Trade-Offs for a Communication Standard in a Data-Driven and Interconnected Aircraft Cabin. *SAE International Journal of Advances and Current Practices in Mobility* 3, 2021-01-0011 (2021), 1197–1205.

[15] Fabian Maximilian Giertzsch, Hartmut Hintze, Burkhard Heinke, and Ralf God. 2019. Network Design Criteria to Introduce Data Analytics Within the Aircraft Cabin. In *7th International Workshop on Aircraft System Technologies,(AST 2019)*. Springer, 363–372.

[16] Andreas Hemmert and Andreas Schweiger. 2022. Development of a SysML Profile for Network Configurations in Safety-critical Systems. (2022).

[17] Sebastian JI Herzig, Robert Karban, Gary Brack, Scott B Michaels, Frank Dekens, and Mitchell Troy. 2018. Verifying Interfaces and generating interface control documents for the alignment and phasing subsystem of the Thirty Meter Telescope from a system model in SysML. In *Modeling, Systems Engineering, and Project Management for Astronomy VIII*, Vol. 10705. SPIE, 319–335.

[18] Pascal Hirmer, Uwe Breitenbücher, Ana Cristina Franco da Silva, Kálmán Képes, Bernhard Mitschang, and Matthias Wieland. 2016. Automating the Provisioning and Configuration of Devices in the Internet of Things. *Complex Syst. Informatics Model. Q.* 9 (2016), 28–43.

[19] International Telecommunications Union. 1994. Information technology – Open Systems Interconnection – Basic Reference Model: The basic model.

[20] Barbara H Liskov and Jeannette M Wing. 1994. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 6 (1994), 1811–1841.

[21] Sissy-Linh Nguyen, Lydia Kaiser, and Pascal Lünnemann. 2023. Methodische Implementierung von Digitalen Zwillingen in bestehenden Systemen. *Tag des Systems Engineering 2023: Tagungsband Würzburg, 15.-17. November 2023* 21 (2023), 234.

[22] No Magic, Inc. 2024. Cameo Systems Modeler 2024x. https://docs.nomagic.com/display/CSM2024x/Cameo+Systems+Modeler+Documentation. Accessed: 2024-03-17.

[23] No Magic, Inc. 2024. SysML Plugin user guide - Creating new validation rules. https://docs.nomagic.com/display/SYSMLP2024x/Creating+new+validation+rules. Accessed: 2024-03-23.

[24] Object Management Group. 2014. Object Constraint Language (OCL).

[25] Object Management Group. 2016. Meta Object Facility (MOF).

[26] Object Management Group. 2017. Unified Modeling Language (UML).

[27] Object Management Group. 2019. System Modeling Language (SysML).

[28] Object Management Group. 2023. Risk Analysis and Assessment Modeling Language (RAAML).

[29] Jennifer Rowley. 2007. The wisdom hierarchy: Representations of the DIKW hierarchy. *Journal of information science* 33, 2 (2007), 163–180.

[30] RTCA. 2011. Software Considerations in Airborne Systems and Equipment Certification, DO-178C.

[31] SAE International Recommended Practice. 2010. Guidelines for Development of Civil Aircraft and Systems, SAE Standard ARP4754A. https://doi.org/10.4271/ARP4754A

[32] Ricarda Schüssler, Florian Balduf, and Martin Becker. 2023. Approaching Smart Interface Specifications in the Systems of Systems Context. *Tag des Systems Engineering 2023: Tagungsband Würzburg, 15.-17. November 2023* 21 (2023), 331.

[33] Daniel Schütz, Thomas Aicher, and Birgit Vogel-Heuser. 2017. Automatic generation of shop floor gateway configurations from systems modeling language. In *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 1–8.

[34] Peter M Shames and Marc A Sarrel. 2015. A modeling pattern for layered system interfaces. In *INCOSE International Symposium*, Vol. 25. Wiley Online Library, 914–927.

[35] Sparx Systems Pty Ltd. 2024. Enterprise Architect User Guide - OCL Conformance. https://sparxsystems.com/enterprise_architect_user_guide/16.1/modeling_fundamentals/ocl_conformance_element_relati.html. Accessed: 2024-03-23.