MODEL-BASED DEVELOPMENT OF INTEGRATED MODULAR AVIONICS ARCHITECTURES ON AIRCRAFT-LEVEL

B. Annighöfer[1], E. Kleemann[2], F. Thielecke[1]
[1]Hamburg University of Technology, Nesspriel 5, 21129 Hamburg, Germany
[2]Airbus Operations GmbH, Kreetslag 10, 21129 Hamburg, Germany

Abstract

The architecture of the integrated modular avionics (IMA) of an aircraft is selected based on the requirements of aircraft systems and the aircraft structure. Aircraft systems using IMA require resources in terms of I/O, memory and calculation power. In addition, systems have secondary requirements like minimum reliability or maintainability. The aim of the IMA design process is to find an architecture, which is compliant to the requirements and is optimal owing to quality measures like cost and weight. Since a huge amount of functions are hosted on IMA in current and future aircrafts, the manual process of defining the architecture is complex, error-prone, and time-consuming. To support this process in speed and maturity, a formal model for IMA architectures is developed, which includes all driving requirements and the resulting architecture. In addition, it enables automated validation, and evaluation of architectures. Moreover, an implementation of modeling, validation and evaluation within the Eclipse development environment is presented.

1. INTRODUCTION

Integrated modular avionics (IMA) is the standardization of avionics hardware and software. The introduction of IMA brought benefits to aircraft development process. System functions can be certified incrementally [1], volume and mass could be saved [2]. Defining the IMA architecture for a new aircraft, however, is a complex, non-automated process. The goal of this process is to design the IMA architecture optimally for all systems and the underlying aircraft. The optimal architecture depends on the distribution of sensors and actuators, the aircraft system's requirements and the aircraft's structure, as well as on the available information technology. All peripherals and the system functions have to be mapped to IMA modules, as well as safety requirements have to be met by the architecture. In addition, the design process starts with uncertain information on the systems becoming more precise later during aircraft development.

Therefore, selecting the IMA architecture is a complex iterative process, carried out by experts. The ever increasing amount of function aboard of an aircraft [3] and the increasingly distributed character of future IMA platforms is going to make these issues even more complex in future. Engineering decisions taken often rely on subjective impression, since the complex architectures can hardly be interpenetrated manually. Thus it is hard to evaluate whether defined architectures are valid and optimal, e.g. in terms of cost or maintainability during this phase. Early validation and evaluation of the IMA architecture is, however, desired due to shorter development time. A formal way of modeling the IMA architecture and the underlying requirements could supports the design process in speed and maturity. Errors are prevented by a model structure and automated validation rules. In addition, the capability of computer-aided evaluation and fast re-design of architectures would be given.

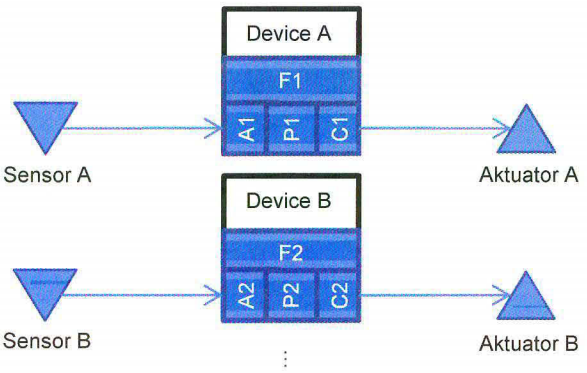
The aim is to develop a computer-aided IMA architecture design methodology. Model-based IMA architecture design shall provide the capability to develop, analyze, optimize and validate IMA architectures during the design phase. In addition, it shall provide the capability to collect architectural requirements from aircraft system departments, and aircraft structure, which can then be used to validate the IMA architecture. The result should be a formal model which includes every attribute of an IMA architecture as required in the scope of the IMA architecture design process. In addition, it shall enable automated calculation of architecture quality measures [4], and support the architecture definition process by enabling fast re-design of the architecture in case of new and more precise input data or changing system requirements. Therefore, IMA and its development process have to be understood to derive model objects, their attributes, and the relationship.

Model-based development is not new in the IMA domain. The Architecture Analysis and Design Language (AADL) [5] is a SAE standard for modeling electronic systems down to the level of processors, as well as software layers and mapping. The Topcased Project [6] provides an implementation of AADL. Gamatie et. all developed a model for IMA software simulations [7] based on the signal language. However, there is currently no approach of rigid modeling the static resource distribution and all additional constraints of IMA architectures on aircraft level. Therefore, a model and methodology for designing, validating and evaluation IMA architectures has been developed.

This article is organized as follows. Chapter two is an introduction into IMA. In addition, it defines IMA architecture in the scope of this article and provides more details on important components and their relation. Chapter three is about the IMA development process. In chapter four a mathematical model for IMA architectures is presented. This is used in chapter five to derive a consistent formal software class model for IMA

architectures. Chapter six is about the technical implementation of the model, its validation and evaluation. Chapter seven summarizes the results and gives an outlook on upcoming research activities.

2. IMA - INTEGRATED MODULAR AVIONICS



(a) Federated Avionics

(b) Integrated Modular Avionics

FIGURE 1. Federated avionics vs. IMA. Key element is the centralized provision of resources by standardized modules (F=Firmeware/OS, A=Acquisition, P=Processing, C=Command).

IMA is the provision of generalized hardware platforms for calculation or I/O task of aircraft systems. These generalized hardware modules are shared between independent applications. Applications originate from different aircraft systems and can be developed independently. Standardized modules ease maintenance, and resource sharing saves hardware and, therefore, weight [8, 9].

Today's IMA first generation modules provide both computation power and I/O interfaces combined in one module. For application development the standardized ARINC 653 API [10] is provided, which is mandatory for all applications hosted on IMA. This API makes I/O operations and data communication between modules transparent for the application developer. This generalization and standardization makes aircraft functions independent of specific hardware. [11]

The currently developed second generation of IMA is going to be more distributed by separating I/O from calculation resources and by allowing a more distributed installation within the aircraft. In addition, reconfiguration in cases failure is addressed. [12]

The development of an IMA avionics system takes place in two areas. One area is the hardware platform defined by the IMA department, consisting of IMA modules and their interconnection. The second area is the systems layer, which includes the functions, which have to be realized in the aircraft by specifying signal flows, signal processing, sources and sinks. Combining both layers is carried out in the integration phase. [11]

In the following basic terms and concepts from the field of IMA as well as are explained.

Platform: The IMA platform is the IMA hardware used in the aircraft. The platform defines the types and amount of IMA modules used as well as their interconnection topology.

Modules: Modules are stand-alone hardware boxes in IMA. A module is the smallest unit in of an IMA hardware platform, e.g. an A380 Core Processing Input/Output Module (CPIOM). In general a module provides resources, which are shared by hosted applications. A resource is a quantity provided for sharing. Resources in current IMA modules are:

- Memory
- Calculation power
- I/O

The resource types might change for future module types. Differences in the provided type or amount of resources define different module types. One module type could provide different interfaces or more calculation power than another. For a generic IMA module it is not specified, which types and amounts of resources have to be provided. No current aircraft is equipped with only one type of module. [8, 2]

ADCN - Aircraft Data Communication Network: The interconnection between IMA modules is standardized and called Aircraft Data Communication Network (ADCN). In current IMA architectures the ADCN is realized as AFDX network. AFDX is a real-time bus based on Ethernet [13, 14]. The ADCN is used for inter-module communication.

System: An aircraft system groups certain functionality, e.g. fire protection. Systems are developed usually in independent departments. Those define the functions to be carried out and the required peripherals. Each system defines locations where peripheral are installed. An overview on aircraft systems and their functions can be found in [9].

Function: Every system includes one or more functions. A function is an encapsulated part of a system. In general it is described by a sensor input, a calculation, and an actuator output. The calculation part is separated in one or more applications.

Application: Applications are software programs executed on IMA modules. In often one function can be realized with one application. However, some functions consist of separated applications owing to reliability or safety requirements, which prohibit running the whole function on one IMA module. In case a function would need more resources then a single module provides splitting a function in several applications is a solution. Applications connect to other applications or its peripherals on abstract software ports. Therefore, an application can be hosted on every module in the architecture providing enough resources. The routing of data to other modules or peripheral interfaces is managed by the A653 OS and is transparent for the application.

Sensors and Actuators – Peripherals: Sensors and actuators are not part of IMA itself. They connect each system to the real world. From an IMA point of view they are peripherals without information on their detailed function. Peripherals, however, play an important role when developing an IMA architecture since I/O interfaces and installation location of peripherals define I/O types and amount needed on IMA modules, as well as cable routes and length. Most peripherals belong solely to a specific aircraft system. There are, however, peripherals shared by systems.

Architecture: An IMA architecture is the combination of the IMA hardware platform, the system applications, signals and peripherals, and the aircraft. It includes all shared resources and their allocation by software and sensors. In addition, it defines the installation of IMA modules and cables within the aircraft.

3. IMA ARCHITECTURE DEVELOPMENT PROCESS

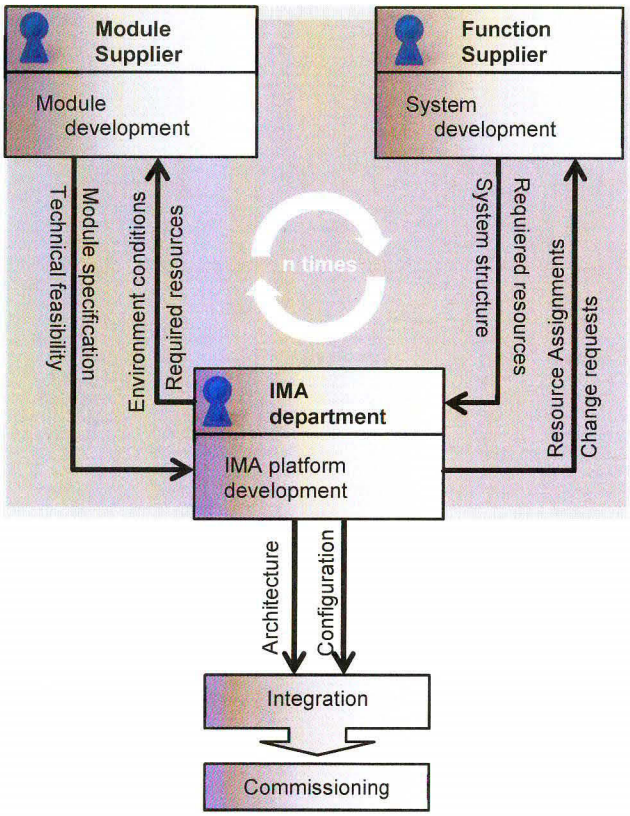


FIGURE 2. IMA development process, participating parties, and exchanged information (cf. [15])

The IMA architecture development process (see figure 2) is a highly concurrent process accompanying the whole aircraft development process. The result of this process is the aircrafts IMA architecture. The IMA department starts with collecting the requirements of systems and the aircraft itself. Meanwhile system functions are developed concurrently by several function suppliers. Based on the information gathered from the function suppliers a first IMA architecture is derived, which provides sufficient resources for all aircraft functions and holds secondary constraints. Since system's functions development starts at the same time, the first requirements are often incomplete and change later in the development process. IMA modules as specified by the architecture are developed by module suppliers. They report the IMA department on technical feasibility. With the results from the architecture design and the module suppliers, a new iteration with the function suppliers begins, where system requirements are refined and the IMA department may communicate design recommendations. Owing to the concurrency the process steps of collecting requirements, creating the IMA architecture, and defining the IMA modules are carried out several times, and the final IMA architecture is the final product of this cyclic iterations.

4. MODELING IMA ARCHITECTURES

An IMA architecture is a distributed system with statically shared resources. Main task during the design phase is to find an architecture, which provides the resources required by the systems and does not exceed the "resources" provided by the aircrafts installation.

This section derives a mathematical model to describe the IMA architecture and its driving requirements from system and installation level.

4.1. System Constraints

From the IMA point of view each system is composed of software blocks exchanging signals and controlling peripherals. Software blocks are i.e. controller, or data acquisition applications. Those blocks are black boxes and, therefore, atomic for the IMA architect. Those atomic software parts are called **tasks** in the following. The set of all tasks T has to be assigned to the IMA hardware. Each task T_i requires a set of certain resources $\mathrm{r}^{\mathrm{T_i}}$ in a certain amount $\mathrm{r}_{\mathrm{j}}^{\mathrm{T_i}} \in \mathbb{R}_+$.

(1) $$r^{T_i} = \left(r_1^{T_i}, \ldots, r_n^{T_i}\right)$$

A **resource** is an abstract unit of what has to be provided to the task and is consumable, e.g. computational power or pins of a certain type of I/O. IMA modules, called **devices**, have to provide resources. An assignment of tasks to a device D_j is valid as long as the device resources $\mathrm{r}^{\mathrm{D_j}}$ are not exceeded.

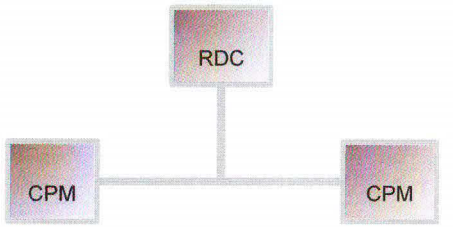
(2) $$\sum_{T_i \subseteq T} r^{T_i} \leq r^{D_j}$$

This must hold for all devices of the architecture D. Alternative resource requirements are, however, possible, i.e. a task can be executed on the same device using different resource sets. E.g. this is true for I/O resources with lightning protection levels (LSP). A sensor requires a minimal LSP, but can also be connected to an I/O with a higher LSP. Therefore, **capabilities** C are introduced. A capability C_k defines a triple of the resource usage $\mathrm{r}_{\mathrm{k}}^{\mathrm{T}}$ of a task T_i on a device D_j. This provides, in addition, a more detailed modeling of which device is possible to execute which task.

(3) $$C_k^{i,j} = \left(D_j, T_i, r_k^{T_i}\right)$$

Signals exchanged between the tasks of the systems can be expressed analogous to tasks. A signal S_i equals a task requiring bandwidth as resource instead of I/Os or computational power. All signals S, however, needs to be mapped to a set of devices D and links L between the devices. Capabilities for signal routing C^S are defined equally to (3).

	r1	r2	r3
Pressure	0	0	1
Valve Left	0	1	0
Valve Right	0	1	0
Resources	***20***	***5***	***2***



	r1	r2	r3
Controller	10	0	0
Monitor	10	0	0
Resources	***90***	***0***	***0***

	r1	r2	r3
Controller	10	0	0
Monitor	10	0	0
Resources	***90***	***0***	***0***

FIGURE 3. Example of resource sharing model. Three devices provide the capabilities to run five tasks under certain resource consumptions (thin rows) and provide certain resources (bold rows). Resource types are e.g. r1=computing power, r2=analogue I/O, r3=discrete I/O.

In addition, to resources there is a need to constrain task distribution by secondary requirements, originating from safety consideration, law regulations or system design traditions. Major secondary constraints are

- peripheral constraints,
- segregation constraints,
- atomic constraints,
- latency constraints,
- device constraints,
- installation location constraints, and
- power constraints.

A peripheral P_i is non-configurable, non-IMA equipment belonging to a certain system. **Peripheral constraints** express that a task requires the specified peripheral in order to be operational. Important for the evaluation of the architecture is the weight of the required cable. Therefore, a peripheral constraint $\mathcal{P}$ includes a tuple of the required peripheral P_j and the cable weight per length $w_l \in \mathbb{R}_+$.

$$\mathcal{P} = (T_j, P_i, w_l), \qquad P_i \in P \tag{4}$$

A **segregation constraint** $\mathcal{S}$ is a tuple of two task T_i and T_j that are not allowed to be executed on the same device, e.g. a controller and a monitor application.

$$\mathcal{S} = (T_i, T_j), \qquad T_i, T_j \in T \tag{5}$$

Moreover, three other types of segregation constraints exist, which define that the two task needs to be executed on dissimilar hardware $\mathcal{S}^D$, in dissimilar installation locations $\mathcal{S}^I$ or both $\mathcal{S}^{D,I}$.

An **atomic constraint** $\mathcal{A}$ defines a set of task $T_{\mathcal{A}}$ that has to be executed on the same device.

$$\mathcal{A} = T_{\mathcal{A}}, \qquad T_{\mathcal{A}} \subseteq T \tag{6}$$

Latency constraints $\mathcal{L}$ define a maximal execution time $t_{\max} \in \mathbb{R}_+$ and a path of tasks $T_{\mathcal{L}}$ and signals $S_{\mathcal{L}}$.

$$\mathcal{L} = (\mathrm{T}_{\mathcal{L}}, \mathrm{S}_{\mathcal{L}}, \mathrm{t}_{\max}), \qquad \mathrm{T}_{\mathcal{L}} \subseteq T, S_{\mathcal{L}} \subseteq S \tag{7}$$

The execution of the specified path $(\mathrm{T}_{\mathcal{L}}, \mathrm{S}_{\mathcal{L}})$ in the final architecture must last below $t_{\max}$. In order to validate latency constraints capabilities are extended by the worst case execution time (WCET) $t_{\mathrm{wcet}} \in \mathbb{R}_+$ a task has on the specified device.

$$\sum_{T_i \in \mathrm{T}_{\mathcal{L}}} \mathrm{wcet}(\mathrm{T_i}) + \sum_{S_i \in S_{\mathcal{L}}} \mathrm{wcet}(\mathrm{S_i}) \leq \mathrm{t_{max}} \tag{8}$$

Where $wcet(x)$denotes the WCET time of the task or signal mapped to a device.

Two types of **device constraints** $\mathcal{D}$ and $\overline{\mathcal{D}}$ exist. Either a task T_i is only allowed to be executed by a set of devices $D_{\mathcal{D}}$ or a set of devices $D_{\overline{\mathcal{D}}}$ is strictly excluded from the possible mapping locations.

$$\mathcal{D} = (T_i, D_{\mathcal{D}}), \qquad D_{\mathcal{D}} \subseteq D \tag{9}$$

$$\overline{\mathcal{D}} = (T_i, D_{\overline{\mathcal{D}}}), \qquad D_{\overline{\mathcal{D}}} \subseteq D \tag{10}$$

Installation location constraint types $\mathcal{I}$ and $\bar{\mathcal{I}}$ are similar to device constraints. Either a task T_i is required to be hosted on a device in a set of certain installation locations $I_{\mathcal{I}}$ or it is prohibited in a set of locations $I_{\bar{\mathcal{I}}}$.

$$\mathcal{I} = (T, I_{\mathcal{I}}), \qquad I_{\mathcal{I}} \subseteq I \tag{11}$$

$$\bar{\mathcal{I}} = (T_i, I_{\bar{\mathcal{I}}}), \qquad I_{\bar{\mathcal{I}}} \subseteq I \tag{12}$$

From safety considerations it can be necessary to specify the power source of a system task. Therefore, two types of **power constraints** $\mathcal{W}$ and $\overline{\mathcal{W}}$ have been derived. From the set of all power supplies W, e.g. DC essential, a task T_i can be constraint to either require certain power supplies $W_{\mathcal{W}}$ or to be prohibited on devices with certain power supplies $W_{\overline{\mathcal{W}}}$. The power source is added as a property of devices.

$$\mathcal{W} = (T_i, W_{\mathcal{W}}), \qquad W_{\mathcal{W}} \subseteq W \tag{13}$$

$$\overline{\mathcal{W}} = (T_i, W_{\overline{\mathcal{W}}}), \qquad W_{\overline{\mathcal{W}}} \subseteq W \tag{14}$$

4.2. Installation Constraints

The IMA architecture to be developed has to be installed in the aircraft. Aircraft installation is on the first hand constrained by the available installation locations I and the available cable routes R between the installation locations. On the second hand installation locations have only limited capacities in space, cooling and power supply. Cable routes are restricted by their diameter. These issues can

be depicted as abstract installation location resources. A device D_i requires a set of installation location resources ρ^{D_i} each of a certain amount $\rho_j^{D_i} \in \mathbb{R}_+$.

(15) $$\rho^{D_i} = \left(\rho_1^{D_i}, \dots, \rho_n^{D_i}\right)$$

Installation location resources are provided by each installation location I_j. A valid assignment of devices to an installation location exist if the sum of the resources required by the assigned devices is smaller or equal to the resources provided by the installation location ρ^{I_j}.

(16) $$\sum_{D_i \subseteq D} \rho^{D_i} \leq \rho^{I_j}$$

The installation location resource concept is also applied to links and cable routes. The resource is e.g. the diameter of the route. Links however are assigned to a set of cable routes, where resource overrun is prohibited in each route individually.

4.3. Mapping

A complete IMA architecture is defined by the elements from the systems, IMA hardware, and aircraft installation layer, as well as the mapping between the layers (s. fig. 4).

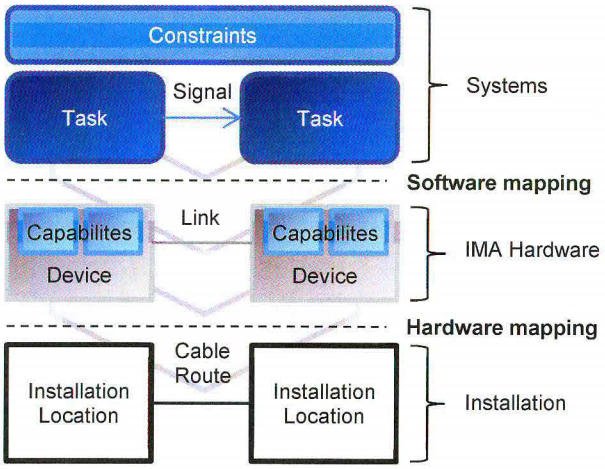


FIGURE 4. Dual mapping of IMA architectures. System software is mapped to IMA devices and devices are mapped to installation locations within the aircraft.

Between the systems layer and the hardware layer the task and signals have to be mapped to devices and links, in respect to all constraints. More precise tasks are mapped to capabilities of the devices. The mapping of all tasks can be expressed in a binary vector $x^K \in \mathbb{B}^n$ where n is the number of all possible assignments of each task within the current hardware setup. A "1" denotes that the task is assigned at the specified location.

(17) $$x^K = (x_1^K, \dots, x_n^K), \qquad x_i^K \in \{0,1\}$$

Within a valid architecture each task must be assigned exactly once.

(18) $$\sum_{1}^{n} x_i^k = |K|$$

Like for tasks an assignment vector x^S is defined for signal assignment. Since signals can have multiple segments (18) is not valid for x^S. However, all assignments of signals must be a valid path in hardware topology.

The mapping between the hardware and installation layer are device assignments and link assignments. Device assignments x^D are expressed like task assignments; also the single assignment constraint (18) is true for devices assignments. Link assignments x^L, however, are expressed like signal assignments. Link assignments must be a valid path within the installation topology.

5. MULTI-LAYER ARCHITECTURE META-MODEL

Based on the mathematical model of IMA architectures an object-based data model (meta-model [16]) for storing IMA architectures has been derived. Besides creating a feasible class structure modularity and reusability have been important requirements for the data model.

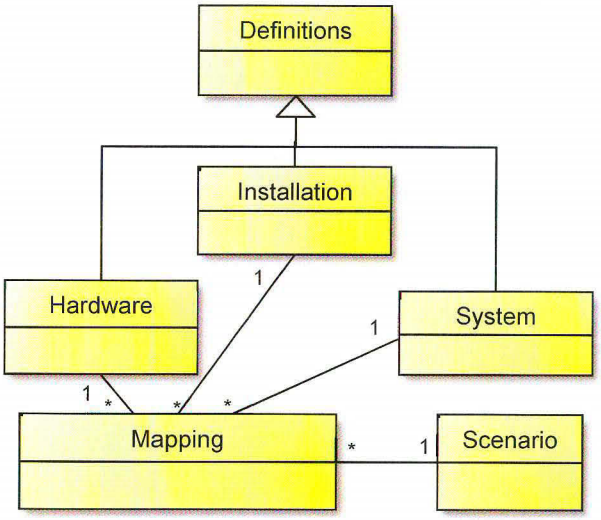


FIGURE 5. Top-level structure of the IMA architecture model. Definitions include basic type definitions for elements of the systems layer, hardware layer, and installation layer. The latter three in addition with scenario are combined to a complete architecture within the mapping layer.

Within this section a data model for IMA architectures is presented, which captures all date relevant for planning, specifying, validating, and evaluating IMA architectures. It provides a resource centered view on IMA architectures. On the top level the model is divided into six layers (s. fig. 4). The **system** layer, the **hardware** layer, the **installation** layer, as well as the **mapping** layer are directly derived from the mathematical model. Since some parameters are necessary for the evaluation of the architecture on aircraft level, which cannot be assigned to one of the previous layers, the **scenario** layer is introduced. Multiple architectures using the same layers as building blocks are

possible within the same model. Moreover, types for the main model elements, e.g. tasks and devices, are specified in **definitions** layer, which increase reusability.

5.1. Definitions Layer

A definition contains a set of basic building blocks for the system, the hardware, and the installation layer. Defined are the types of devices, tasks, etc. In addition, types define properties all instances of a specific type have in common, e.g. weight of a device type. With device types, resource types, and task types the resource based distribution is realized. Each device type can have an amount of resources of a specific type. The capabilities of the device type define the amount of resources consumed by a task type, when hosted by a device of this type. For the early planning phase, where device types are unknown, a task type can specify the resources required, which later on has to be transferred into device type capabilities.

Resources are quantities that are provided by devices and are consumed by task and signals. Each device resource type is identified by a name. In addition, if applicable, a type shall provide a unit by which the amount of resources of this type is counted. E.g. processing power resource could be counted in MIPS.

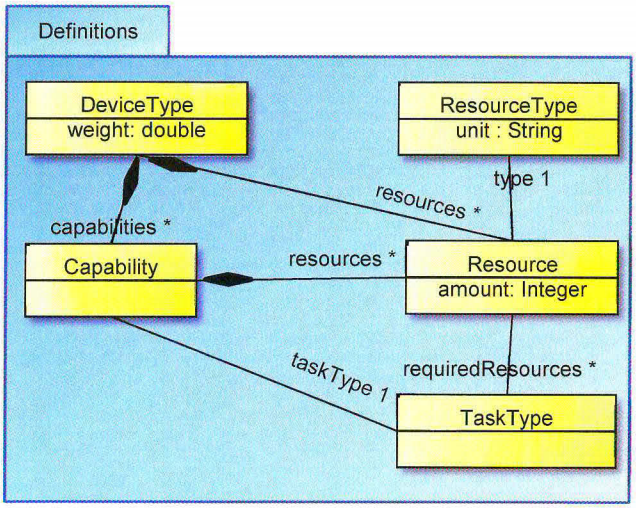


FIGURE 6. Selected elements of the definitions model. Depicted are resource, task, and device types. Task types require resources. Device types provide capabilities under a certain resource consumption.

5.2. Systems (Software) Layer

System models define the logical structure and requirements of system functions to be executed on the IMA architecture. Most important elements are tasks, signals, peripherals, and constraints.

Tasks are arbitrary atomic pieces of software, which are executable on IMA hardware. E. g. a controller application for a Core Processing Module (CPM) is a task as well as the reading and routing of an digital input signal to an AFDX message on an Remote Date Concentrator (RDC) is a task. Each task references to a certain task type of the definition model. The task type defines on which devices the task is executable and which resources are consumed. Two task of the same type in the same software model are valid, e.g. for the reading data from two sensors of the same type. The level of detail when modeling tasks is up to the user. E.g. a task can represent a single logical function or a complete controller with sensor and actuator drivers. However, the level of detail defines the distribution possibilities of a system. One task cannot be distributed on two devices. The replacement of this task with two tasks and one signal in between, however, can be distributed. Task groups and task sets enable structured modeling and multi-task constraints.

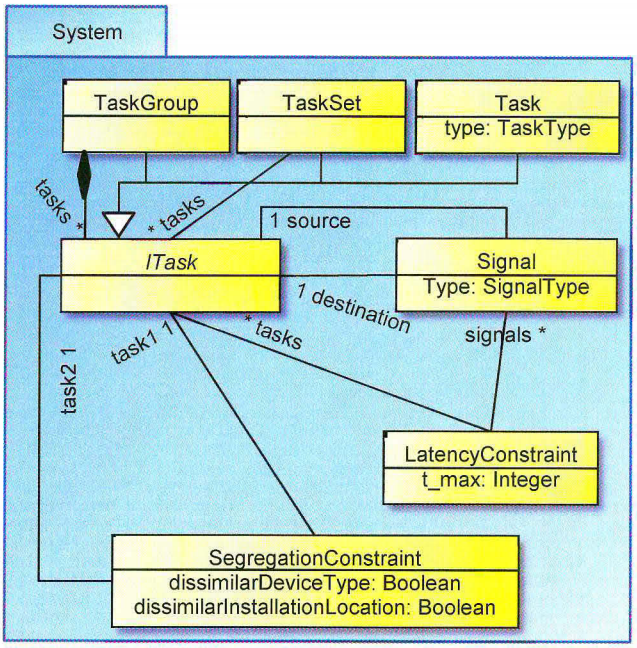


FIGURE 7. Selected elements of the systems model. Depicted elements are tasks, signals, peripherals, segregation and latency constraints.

Signals define the data exchange between two tasks. A signal references a signal type that defines its basic properties and possible mapping targets. A signal is a data exchange from one task defined as source to the task defined as target. More than one signal between the same tasks is allowed. Like tasks the level of detail when modeling signals is chosen by the user. E.g. a signal can represent the full data exchange between two tasks or each exchanged primitive data type can by represented by one signal. The higher the amount of signals the higher is the amount of distribution possibilities in the mapping.

Peripherals represent instance of hardware not belonging to the set of IMA hardware. These are sensor or actuators in general. The reason why peripherals are modeled in the software domain is that unless binding a certain task that requires the peripheral to a device there exists no connection between an IMA device and a peripheral. The logical connection of peripherals and task, however, is mandatory and settled. Moreover, function supplier will know the kind of sensor and its location in the installation better then IMA hardware designers.

Since topology and resources are not sufficient to describe all requirements, additional constraints can be defined. Constraints are assigned to task, task groups, and task sets.

A Segregation Constraint references to tasks or signals that are not allowed to be executed or routed on the same

device or link. Moreover, two additional attributes allow more strict segregation.

- *Dissimilar Device Type*: If this attribute is set to true, it means that the task or signal is not only restricted to different devices or links, but also to devices or links of a different type.
- *Dissimilar Installation Location*: If this attribute is set to true, it means that the task or signal is not only restricted to different devices or links, but also to devices or links installed in different installation locations.

Latency Constraints define a worst case execution time (WCET) for the signal flow and task execution between both endpoints of the constraint. A latency constraint is hold if the sum of the WCET for each part of the signal processing chain is below or equal to the time specified with the constraints. WCET of a task execution or signal transportation is defined within the capabilities of device types and signal types.

Device Constraints define a set of devices the task, task group or set must or must not be mapped to.

Installation Location Constraints define a set of installation locations the device of the task must or must not be mapped to.

Power constraints define a set of power supply types the device of the task must or must not be connected to.

Reliability Constraints define the minimum mean-time-between-failure (MTBF) for the tasks after the mapping.

Task of the task group or set assigned with an Atomic Constraint must always be mapped to the same device.

5.3. Hardware layer

The hardware layer contains the topology of the IMA hardware for the IMA architecture. It includes devices and there interconnection by links. Although the type of links can be arbitrary in most cases the set of links can be seen as the ADCN network.

Devices are instances of a device type from the definition model. The device type of a device is defined by a reference to the device type object. A device is identified by its name. Devices are instances where task and signals from the software model can be mapped.

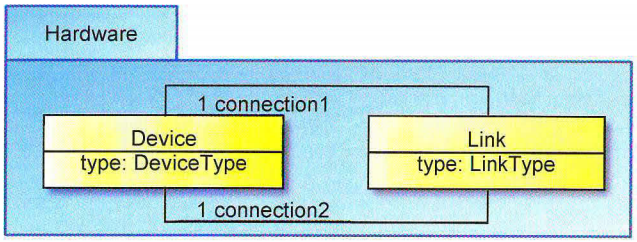


FIGURE 8. Selected elements of the hardware model. Depicted are the devices and their link-based communication structure.

Links are defined by the link types from the definition model. Signals from the software model can be mapped to links. Links are identified by their name. In addition, each link has two references (connection1 and connection2) to devices. This enables direct data exchange between the referenced devices.

5.4. Installation Layer

The installation model includes possible installation locations and cable routes within the physical dimensions of the aircraft. Objects of the installation model are mandatory to define the mapping of the hardware topology to the aircraft and to calculate cable lengths between mapped module and peripherals.

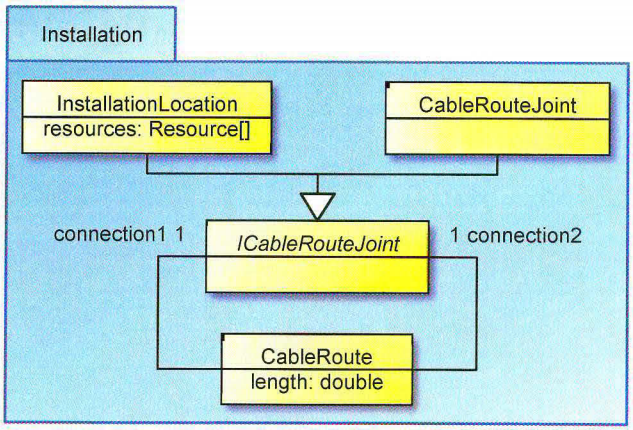


FIGURE 9. Selected elements of the installation model. Depicted elements are installation locations and provided resources, as well as cable routing paths.

Installation Locations are places, where modules from the hardware model or peripherals from the system model can be installed. Therefore, each installation location contains installation location resources it provides, e.g. space, power or cooling capacity. For each installation location resource the type and the provided amount is defined. Each installation location itself has an absolute location in a physical global aircraft coordination system.

Cable Routes are connections between installation location and/or cable root joints. A cable route is assigned with a fixed length independent of the position of the connected objects. Cable routes can be assigned with links from the hardware model or peripheral connections from the software model.

Cable Route Joints provide the possibility to split or merge two cable routes. This way not every point-to-point connection has to be an individual cable route object.

5.5. Scenario Layer

A scenario includes properties that are important for quality measure calculations but may change with the usage scenario of the aircraft. Often the attributes of the scenario are values defined by the customer of the aircraft manufacturer. Important business constants are e.g. number of seats, seat usage, delay cost per hour per seat, cancellation cost per seat, and annual usage.

5.6. Mapping Layer

The mapping model includes mapping objects that assign system software to hardware and hardware to aircraft installation locations.

5.6.1. Software Mappings

Task assignment (s. fig. 9) objects represent the mapping of a software task to a device from the hardware. Each task assignment object consists of a reference to the

device the task is mapped to, a reference to a task, and a reference to the capability of the device type the task uses to run on the device.

Signal assignments define, in analogy to task assignments, the mapping of signals from the system model to links and devices from the hardware. Signals are mapped to links for inter-device communication and to devices for intra-device communication. Therefore, signal assignment objects contain a link to a signal and n Signal Segment Assignments objects, where each contains a link to a device or link, and a reference to a capability of the used device or link type. The latter defines the resource usage by signals. Signal Segment Assignments shall be set in such a way that modules and links the signal is assigned to are connected in the hardware device, and that the form a valid path.

5.6.2. Hardware Mappings

Device assignment objects (s. fig. 9) define the physical location of a device from the hardware topology by mapping it to an installation location of the aircraft. These assignments define the usage of installation location resources.

Link Assignments define the mapping of links from the hardware topology to cable routes in the aircraft. Each assignment object contains a link to a link object and n links to cable routes. A cable route can be either a physical cable route or a cable route joint or an installation location, which has to be passed. A valid link assignment only contains links to cable routes which are connected in the aircraft model.

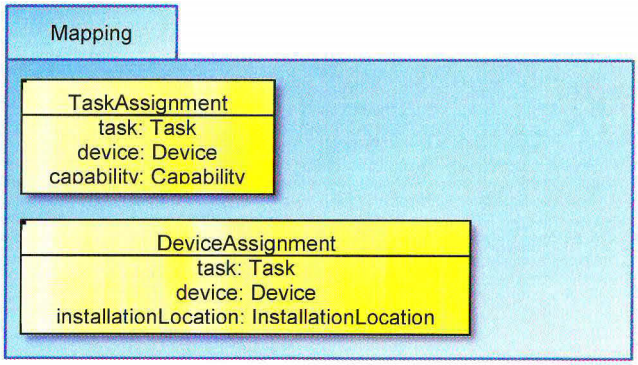


FIGURE 10. Selected elements of the mapping model. Tasks are mapped to devices and devices are mapped to installation locations.

6. SOFTWARE IMPLEMENTATION

The presented data model for IMA architectures has been implemented using a model-based development process utilizing the Eclipses Framework [17] and its domain specific modeling extensions the Eclipse Modeling Framework (EMF) [18] and the Eclipse Graphical Modeling Framework (GMF) [19]. In addition, a detailed validation and several quality measures have been implemented. As depicted in figure 11 the implementation is modularly separated in the model, a user interface for editing the model, a generic validation component, and an evaluation component.

This section describes the techniques and methodology used to implement an IMA architecture development environment and provides feedback on the utilized frameworks.

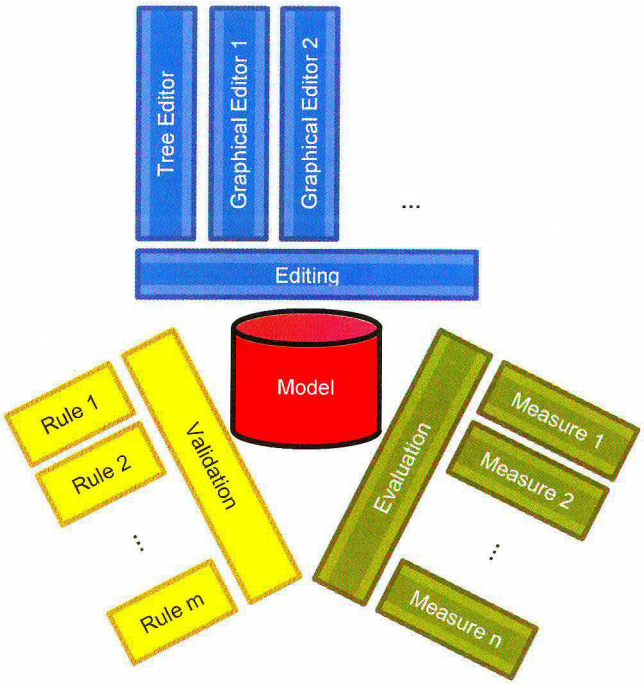


FIGURE 11. Modular software architecture of IMA architecture development environment. Editing, validation, and evaluation are centered around the data model. Each of the three model interfaces can be modularly extended by additional validation rules, quality measures, and editors.

6.1. Model – Data Handling

The model implementation is the storage of the IMA architecture data as specified in in section 5. The model has been implemented using the EMF Ecore modeling language, which is an implementation of the Essential Meta-Object Facility (EMOF) [20] standard. Using Ecore the data models are composed of classes, relationships, containments, and attributes. From the Ecore model Java code has been generated using the EMF code generator. The generated model code can be extended by custom code and can be updated by later model changes. The implementation of the model includes meta-model information during runtime. The model can be extended offline and online. In addition, it includes loading and saving routines to the XML Metadata Interchange (XMI) [21] standardized file format. Moreover, the generated implementation serializes access to the model data and provides change listeners on model elements.

The used development methodology eases the implementation of complex domain specific models and leads to a state-of-the-art implementation of the model.

6.2. Editing

An editing interface which provides access to the model is generated with EMF. In addition, a tree-like editor for the domain specific model has been generated with EMF. This editor represents the plain model structure and allows editing of all model features. The tree-view, however, in general does not visualize what should originally be

represented by the model, e.g. an IMA hardware topology. Therefore, Eclipse GMF has been used to develop more advanced graphical 2D diagram editors. Those diagram editors display model content as 2D shapes and connections between them. GMF again is based on model-based code generation. The design, layout, and behavior each model element is described in GMF modeling languages. From those models GMF generates code for a diagram editor, which can be extended by custom modifications.

GMF has been used to generate a diagram view for each top level layer of the IMA architectures model, e.g. system (s. fig. 12) and hardware (s. fig. 13). Owing to the EMF editing interface all editors can be used concurrently on the same data. The five resulting diagram views are nested, i.e. one diagram view can be opened from another by clicking on the objects related to a diagram. Diagram view information is stored separated from the model, such that multiple views on the same architecture model are possible.

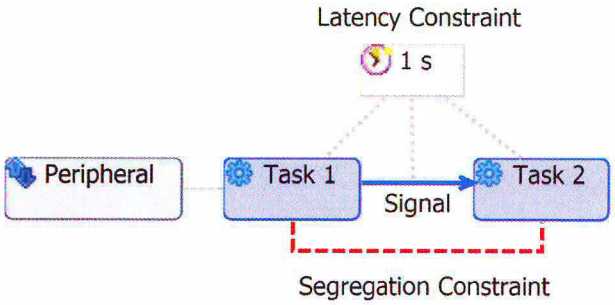


FIGURE 12. Systems model example diagram

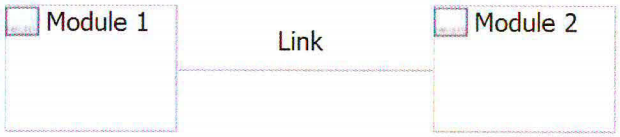


FIGURE 13. Hardware model example diagram

GMF provides a fast way creating diagram editors with, state-of-the-art features. Complex model structures, e.g. indirect relationships, which occur e.g. in the mapping layer, can currently not be sufficiently visualized with GMF.

6.3. Validation

The structured modeling approach allows inherent validation of basic modeling errors, e.g. empty attributes. Complex validation rules, like a check for resource exceeding, however, are not covered by Ecore. Therefore, the Validation Framework included in EMF has been utilized. The validation framework allows specifying arbitrary validation rules in Java.

The validation of IMA architecture constraints has successfully been implemented using the Validation Framework. Validation can be done during editing and error sources can be located by icons easily. Additional validation rules can be added using the Eclipse plug-in mechanism.

6.4. Evaluation

During the design different quality measures shall be calculated for the current architecture. Therefore, a quality measure registry has been implemented, which enables the registration of arbitrary quality measures for certain model object classes. If feasible the registered measures are calculated on-the-fly, when the user selects an object of the type. In addition, it is recalculated if a dependent attribute in the model changes. Measure results are shown to the user and can be used programmatically. To avoid frequent recalculation measure results are buffered.

This way the IMA architectures are evaluated owing to the implemented quality measures during design. In addition, the modular implementation enables the addition of arbitrary quality measures as separate Eclipse plug-ins later on.

7. CONCLUSION AND OUTLOOK

IMA is a standardization approach in the field of avionics. Sharing resources of standardized hardware modules between the aircraft systems lower weight and maintenance cost. This article provides an introduction into IMA and the related process. It shows that the rising system complexity and contradictive quality measures, as well as the high degree of freedom make it difficult to create the optimal architecture manually. Model-based design of IMA architectures shall support the process, by formally collecting requirements of the aircraft systems and the aircraft structure, and by validating and evaluating those during the design of the IMA architecture.

A mathematical model of IMA architectures has been derived using set and graph theory. Most important, the concept of a task has been derived, which is an atomic software unit to be distributed on the IMA hardware. Tasks are constrained by their resource needs and secondary constraints originating from safety and regulations, e.g. segregation or power source constraints.

From the mathematical a formal software class model has been derived. This domain specific model defines IMA architectures in the four basic layers, system, hardware, installation, and mapping, and two supporting layers for type definitions and scenario global parameters. System contains the logical system structure, its periphery and constraints, hardware is an IMA hardware topology, installation contains installation locations and cable routes, and the mapping layer defines the final IMA architecture by assigning software to hardware and hardware to the aircraft. The design of the model enables modular and reusable design of IMA architectures by combining layers.

The IMA architectures class model has been implemented using the Eclipse Modeling Framework (EMF). This open source model-based software engineering approach enabled semi-automatic generation of state-of-the-art model implementation and user-interface. Moreover the experience using EMF, GMF and its companions has been documented, e.g. difficulties of creating graphical views of complex model structures. The created development environment for IMA architectures is a modular composition of editing, validation, and evaluation centered around the data model. A novel quality measure registry allows live evaluation of arbitrary measures on arbitrary model objects. EMF Validation Framework has been used to implement validation rules not validated by the model structure itself. Editing, validation, and quality measures are inherently extensible by Eclipse plug-ins.

A mathematical and data model of IMA architectures are first steps necessary for computer-aided IMA architecture

design. Having those available enables new possibilities in two directions, which will be investigated within further research activities.

First, the machine readable information can be used for algorithmic architecture optimization, e.g. finding the optimal software distribution on an IMA hardware.

Second, the formal data model and the Eclipse Framework enable automatic data transitions from or to other phases or departments of the aircraft development process which could make the process faster and less error-prone. For example formal model transformation technics could be utilized the transfer information from IMA architecture data to IMA configuration data.

8. REFERENCES

[1] René L.C. Eveleens. Integrated Modular Avionics Development Guidance and Certification Considerations. http://ftp.rta.nato.int/public//-PubFullText/RTO/EN/RTO-EN-SCI-176///EN-SCI-176-04.pdf, November 2006.

[2] P.J. Prisaznuk. Integrated Modular Avionics. In *Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National*, pages 39–45 vol.1, May 1992.

[3] Jean-Bernard Itier. IMA1G - Genesis and Results. SCARLETT MOSCOW - 1st Forum, September 2009.

[4] K. Neumann, Ernst Kleemann, R. Reichel, and M. Lehmann. Quantitative Evaluation Criteria for Modern Avionic System Architectures. In *Deutscher Luft- und Raumfahrtkongress*. DGLR, September 2008.

[5] SAE. Architecture Analysis & Design Language (AADL), January 2009.

[6] Topcased. The Open-Source Toolkit for Critical Systems. http://www.topcased.org/, 2007.

[7] A. Gamatie, C. Brunette, R. Delamare, T. Gautier, and J.-P. Talpin. A Modeling Paradigm for Integrated Modular Avionics Design. In *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, Pages 134–143, 29 2006-Sept. 1 2006.

[8] Cary R. Spitzer. *Digital Avionics Handbook*. CRC Press, 2007.

[9] Ian Moir and Allan Seabridge. *Aircraft Systems*, volume 3. Wiley, 2008.

[10] Wind River Inc. Wind River Systems / IEEE Seminar. http://www.computersociety.it/wp-content/uploads/-2008/08/ieee-cc-arinc653_final.pdf, August 2008.

[11] Reiner Gnauert. Dynamische Re-Konfiguration im Redundanzmanagement fehlertoleranter Flugzeugsysteme auf Basis von IMA. Diplomarbeit, Technische Universität Hamburg-Harburg, June 2004.

[12] Didier Hainaut. SCARLETT - Towards the next generation of Integrated Modular Avionics. European and Russian Joint Avionics Forum - New Generation of IMA Solutions for Future Aircraft, September 2009.

[13] GE Fanuc Embedded Systems. AFDX/ARINC 664 Protocol Tutorial, 2007.

[14] F. Brajou and P. Ricco. The Airbus A380 - An AFDX-based Flight Test Computer Concept. In *AUTOTESTCON 2004. Proceedings*, pages 460–463, Sept. 2004.

[15] Martin Halle and Frank Thielecke. Konfigurationsmanagement für Integrierte Modulare Avionik. In *Deutscher Luft- und Raumfahrtkongress, Hamburg 31. August - 2. Sept. 2010*, Number 1213. Deutsche Gesellschaft für Luft- und Raumfahrt, September 2010.

[16] D. Karagianis and H Kühn. Metamodelling Platforms. In *Proceedings of the Third International Conference EC-Web*, Number 2455 in LNCS, Page 182, Aix-en-Provonce, France, September 2002. Springer-Verlag Berlin Heidelberg.

[17] Eclipse. http://www.eclipse.org/, November 2009.

[18] Eclipse. Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/.

[19] The Eclipse Foundation. Graphical Modeling Project (GMP). http://www.eclipse.org/modeling/gmp/.

[20] Meta Object Facility (MOF) Core Specification. http://-www.omg.org/spec/MOF/2.0/PDF/, January 2006.

[21] Xml Metadata Interchange Specification. http://-www.omg.org/spec/XMI/, July 2005.