# HARDWARE-SOFTWARE ALLOCATION SPECIFICATION OF IMA SYSTEMS FOR EARLY SIMULATION

*Ákos Horváth, Ábel Hegedüs, Márton Búr, Dániel Varró, Budapest University of Technology and Economics, Department of Measurement and Information, Budapest, Hungary*

*Rodrigo R. Starr, Samoel Mirachi, EMBRAER, Sao Jose dos Campos, SP, Brazil*

## Abstract

Model-driven engineering (MDE) is becoming a key approach in systems engineering, including Integrated Modular Avionics (IMA) design. It relies on systematic use of models from an early phase of the design process to provide source code generation, validation and analysis support. However, due to the complexity of IMA systems – that may incorporate hundreds of avionics functions and dozens of execution nodes – even early stage model-based analysis of their design can become cumbersome. This is especially true for safety related non-functional requirements like communication channel redundancy or error propagation and contamination.

In this paper, we present a model-driven framework to support the iterative design and analysis of IMA systems using an integrated Simulink model for analyzing the complete HW-SW architecture of the system.

## Introduction

Modern Integrated Modular Avionics systems bring a lot of flexibility to avionics systems development, but with this flexibility comes a more challenging design process for precisely configuring its hardware-software execution platform. This significantly raises the complexity of IMA system design compared to federated architectures, where the application software is statically allocated to its execution hardware.

Within IMA each possible configuration has to fulfill several different functional and non-functional requirements (e.g. safety, bus capacities, timing), with some typically unknown at the early stage of design. To overcome this limitation typical system design approaches execute an iterative development process, where the first iterations explore a larger part of the design space to define the boundaries of the system high-level architecture and later iterations focus on low-level details within the defined boundaries. However, due to the complexity of IMA systems, even the early, high-level analysis can take a considerable amount of time and effort especially, when taking into account safety requirements, making it viable economically to invest in tools to automate the definition and analysis of these types of system.

In the current paper, we present the results of the Trans-IMA project – a co-operation between Embraer and the Budapest University of Technology and Economics – that defined a model-driven approach to support the iterative design, refinement and analysis of IMA systems. The approach is based on the automated generation of an integrated Simulink model for analyzing the complete hardware-software architecture of the system using a high-level allocation process for mapping the avionics functions to their executing or implementing hardware platform.

The approach was realized on the Eclipse platform [1] as it provides cutting edge modeling features ranging from model definition to model querying and management.

### *Outline*

In order to introduce our approach we first (i) provide an overview that describes the main artifacts of Trans-IMA and (ii) present a motivating case study. Next, we (iii) list enabling technologies with their main features. After (iv) describing the modeling architecture and (v) the overview of the tooling, we (vi) provide details for each module. Finally, we (vii) highlight related research and (viii) conclude the work.

## Trans-IMA approach

Trans-IMA aims at defining a model-driven framework for the synthesis of complex, integrated Matlab Simulink models capable of simulating the software and hardware architecture of the avionics system of an aircraft. A high-level overview of our approach is depicted in Figure 1.

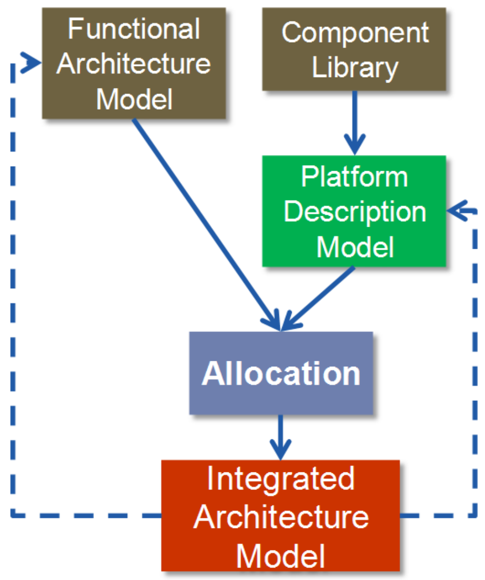
Functional Architecture Model
Component Library
Platform Description Model
Allocation
Integrated Architecture Model

**Figure 1. The Trans-IMA approach**

The avionics functions are defined using a **Functional Architecture Model** (FAM). A FAM is an abstraction of the avionics functions including their functional decomposition and their corresponding information links (the data flow structure) from a Simulink model. This functional model can have a varying level of detail, depending on the phase of the project and the goal of the analysis.

The underlying hardware architecture is defined using a dedicated **Platform Description Model** (PDM). PDM defines a set of generic components (such as routers, processing units, buses, chassis etc.) that can be used to define the overall execution and communication architectures. The internal behavior of these generic components is defined in various Simulink libraries (**Component Libraries**) to support the different simulation goals and provide an extension mechanism for vendor specific hardware elements.

Based on systems defined by a FAM and a PDM the system architect can specify an allocation between the functions and their execution platform. The allocation itself includes two major phases: (i) the mapping of avionics functions to the underlying execution elements and (ii) the automated discovery and selection of available communication paths in the hardware architecture for the various information links defined between the avionics functions. Finally, when the allocation is complete – all functions and information links are allocated – and fulfills the safety and design requirements, an integrated Simulink model – called **Integrated Architecture Model** (IAM) – is automatically generated, where all Simulink specific configurations like bus creators and selectors, library links and model references are configured. This integrated model can then be submitted to a common set of tests and analyses to generate performance figures for each allocation configuration.

This early separation of concerns between the functions and their execution hardware enables two major advantages in the design process: (i) first, *during hardware supplier procurement* it allows *fast evaluation* of the advantages and disadvantages of each proponent's architecture, since the (preliminary) avionics functions are reused and only a new hardware architecture model and allocation have to be done for each proposal. (ii) In later phases of design, when a platform supplier is already selected, typically the hardware architecture is defined, but the *avionics functions may be varying* based on newly occurring derived requirements or function provider requests that require the *rapid reevaluation* of several allocation options. Again, the framework allows reusing already defined allocations between slightly modified functions to the same hardware architecture allowing reallocation only of the changed parts and thus significantly reducing reevaluation time for the different options.

As a summary, our approach applies an abstraction based approach for defining complex IMA hardware-software architectures using model-driven techniques that allows the automated generation of an integrated Simulink representation for analysis and simulation purposes.

## *A motivating case-study*

The motivation behind the Trans-IMA approach can be demonstrated with a simple avionics system example including a functional architecture that may be allocated to two different physical architectures. The example is loosely based on the IMA system of the A380 as described in [2]. The functional model may be inherited from previous programs and allows the study of different physical architectures in the preliminary design phase.

The top-level view of the functional architecture is shown in Figure 2, with sensors and actuators differentiated from functions by color and label. Note that *FADEC Functions* integrates a non-IMA system that can work as a stub for software that is owned by one of the suppliers. Functions may be decomposed into sub-functions to form a hierarchy.

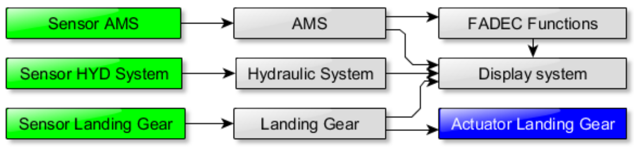

**Figure 2. Functional architecture**

The physical architecture has a chassis configuration as a building block, as shown in Figure 3. The chassis has three *processing units*, each with an *IMA RTOS* and *communication bridge*, powered by two *power supplies* and connected to two *input/output (I/O) modules* through a *databus* in the backplane of the chassis. The I/O modules connect to *AFDX switches* for inter-chassis communication.

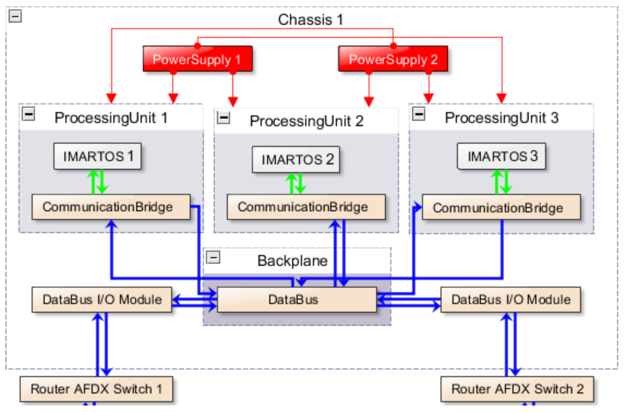

**Figure 3. Detailed chassis platform description**

Two different platform descriptions are studied, with the first variant shown in Figure 4. It contains two chassis with three processing units each, connected by two AFDX switches. In addition, they both connect to the federated unit *FADEC*.

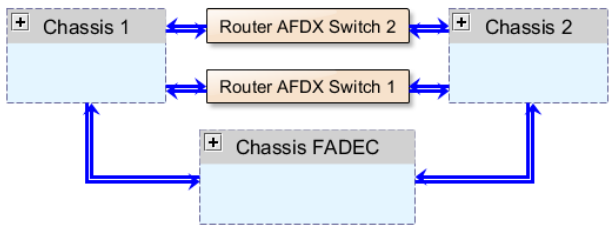

**Figure 4. Platform description, variant I.**

The second platform description variant, shown in Figure 5[1], uses three chassis, each containing only two processing units, with only two of them connected to the FADEC.

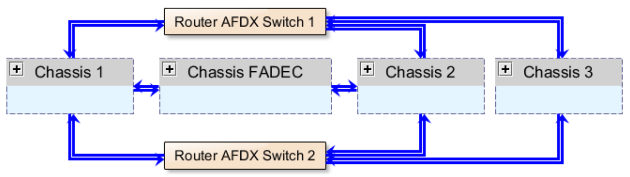

**Figure 5. Platform description, variant II.**

By studying two variants, it is possible to evaluate the trade-offs between placing more processing units into fewer chassis while the addition of another chassis may increase communication overhead in the switches and introduces constraints on the allocation, since the third chassis is not connected to FADEC.

In the Trans-IMA approach, this case study also demonstrates the importance of reusing allocation specifications as details are added to the functional model and when switching between platform variants.

Finally, the example illustrates that to provide usability in addition to functionality, the Trans-IMA tooling has to support hierarchical models and views that focus on specific fragments of the model.

[1] Note that these figures are taken directly from the Trans-IMA Tooling without modification.

## Enabling technologies

MDE relies on two key technologies that allow the definition and manipulation of models, called, metamodeling and model transformation, respectively.

**Metamodeling** is a methodology for the definition of modeling languages. A metamodel specifies the syntax (structure) of a language. Metamodels are expressed using a metamodeling language that itself is a modeling language. The metamodel can also be interpreted as the object-oriented data model of the language under design. There are several different metamodeling environments, most widely used are *the Meta Object Facility* (MOF) [3] from OMG and the *Eclipse Modeling Framework* (EMF) [4] (a subset of MOF).

**Model transformations** (MT) are the backbone of the MDE concept. Primarily, MTs are responsible for transforming the various models into each other. However, MTs can also define *views on models* and *synchronization* between different models (like UML class diagrams and relational database schemas). Moreover, engineering models are frequently mapped into mathematical domains by model transformations to carry out *model analysis* as early model based verification. Well-known approaches for high-level declarative specification of model transformations are the *ATLAS Transformation Language* (ATL) [5], the *VIATRA2* (VIsual Automated model TRAnsformations) system [6] and the *GReAT* (Graph Rewrite And Transformation) framework [7].

The Trans-IMA approach is built on top of the Eclipse platform and integrates several *open source* technologies to support the required use cases.

### *Eclipse modeling framework (EMF)*

EMF is a core Eclipse technology that supports the definition of domain-specific languages (DSLs) through metamodeling and provides a code generation facility that derives interfaces and ready-to-use implementation for a DSL. A large number of Eclipse applications include EMF based DSLs or provide additional features to work with EMF models (e.g. validation, transformation, user interface forms).

The metamodeling language of EMF is called Ecore and it supports the definition of interconnected metamodels (also called Ecore models) thus it is possible to extend and integrate existing DSLs in a straightforward way.

Ecore models are processed by the code generation facility of EMF that also uses a generator model which can be used to customize the generation process. The generated code includes the interfaces and implementation for metamodel classes, several utility classes for processing and creating instance models, additional classes for binding the elements of the models to UI components and finally a fully featured tree view-based editor for viewing, creating and modifying instance models.

From the tool development point of view, one of the main advantages of EMF is that it provides a reflective API that can be used for handling instance models conforming to any metamodel without knowing about the metamodel when the tool is created. Thus generic EMF tools can provide features (e.g. validation, transformation, graph-based views) for any DSL.

In addition, since EMF models are accessed through interfaces, it is possible to replace the implementation without affecting existing applications. For example, the instance models may be read from and stored in a database or derive from the data representation of a legacy application. Finally, EMF provides a command stack based editing functionality which supports transactional access including undoing and redoing model changes.

Thanks to its well-defined core and wide usability, EMF is considered the de facto standard for modeling in the Eclipse ecosystem. The high number of EMF based tools also signifies that it is a mature technology.

### *EMF-IncQuery*

EMF-IncQuery [8] is a model query framework for EMF models which includes a declarative query language [9] based on graph patterns and a very efficient incremental pattern matcher. Model queries are specified in a text editor created using Xtext which is a framework for developing textual domain-specific languages. It is possible to define queries over arbitrary EMF metamodels and then execute them efficiently and incrementally, with proven scalability [10] for complex queries and large instance models (with millions of model elements). Incremental execution is backed by internal caches of partial query results, which are updated based only on model changes without traversing the complete model. The results of a query are always up-to-date and instantly available to the user.

In the query language graph patterns are represented by a set of constraints, where variables can refer to model elements and attributes. Structural constraints prescribe the interconnection of model elements of a given type, while attribute constraints are defined using expressions. A negative application condition (NAC) defines cases when the original pattern is not valid, in the form of a negative sub-pattern. In addition, check constraints allow the execution of pure, deterministic functional methods (with no side-effects).

The incremental pattern matcher uses a Rete-based approach [10] that relies on a network of nodes storing partial matches of patterns. Input nodes represent underlying EMF model elements, intermediate nodes are used to execute basic operations (e.g. filtering, projection or join), while match results are available as an output (or production) node. The input nodes of the network are set up to receive notifications about changes affecting the EMF models and these nodes release update tokens to related intermediate nodes. Rete nodes update their caches based on these tokens and propagate updates through the network, eventually influencing the match results stored in production nodes.

As a distinguishing feature, queries can be integrated into existing applications by processing query results through an easy-to-use API and by taking advantage of extensions [10] for viewers and data binding to user interface components, live validation of constraints, query-based derived features. EMF-IncQuery also supports query libraries which contain reusable definitions that can be integrated into complex queries using the pattern composition concept of the query language.

Each EMF-IncQuery extension is developed to work in an incremental way as well. To provide a common foundation for these extensions, EMF-IncQuery has an event-driven rule engine that can also be used for tool developers and supports both batch and incremental model transformations.

### *Xtend*

Xtend [11] is a programming language that is often called “modernized Java” because it is based on Java, generates Java source code, interoperates with any Java code but includes a large set of improvements over Java, such as type inference, lambda expressions, dispatch methods and template expressions.

Since EMF-IncQuery focuses on model queries and does not have a complete transformation language (contrary to VIATRA2), model transformations in Trans-IMA were developed in Xtend. Both the query engine and event-driven rule engine APIs are well suited for Xtend and we also developed an internal DSL which incorporated the basic functionality required to define event-driven model transformations in Xtend.

Thanks to several language features, we were able to create more concise and maintainable transformations than would have been possible in plain Java.

### *Graph layout and visualization libraries*

The viewer extension of EMF-IncQuery can be integrated with any Java based graph layout and visualization library, we found that none of the open source libraries can scale to the graph sizes that were required for Trans-IMA and more importantly their layout algorithms were unable to work incrementally.

Therefore we chose yFiles for Java [12] to develop Trans-IMA views and integrated it with the viewer extension of EMF-IncQuery, even though yFiles uses AWT for visualization, while Eclipse user interfaces are built on SWT.

The most important features of yFiles for Trans-IMA were the following: (i) incremental layout algorithms scaling to thousands of graph nodes and edges, (ii) hierarchical graphs and closable group nodes and (iii) easy fine-tuning of layout and visualization parameters.

## Modeling architecture

In order to support the envisaged model-driven Trans-IMA approach we defined a set of EMF metamodels to capture all the relevant information from the functional architecture and the hardware component libraries both defined in Matlab Simulink (depicted in Figure 6). The driving idea behind our modeling architecture is to (i) provide a platform (vendor) independent modeling layer for both the functional and hardware architecture allowing future extensions to other simulation and specification languages like AADL [13] and Modelica [14], respectively, (ii) allow instance model reusability for multiple allocation scenarios like single FAM to multiple PDM and vice-versa and finally (iii) define the metamodels from the beginning taking into account traceability through the complete process.

To ease the understanding of the different metamodeling concepts we provide small example models from our running case-study.

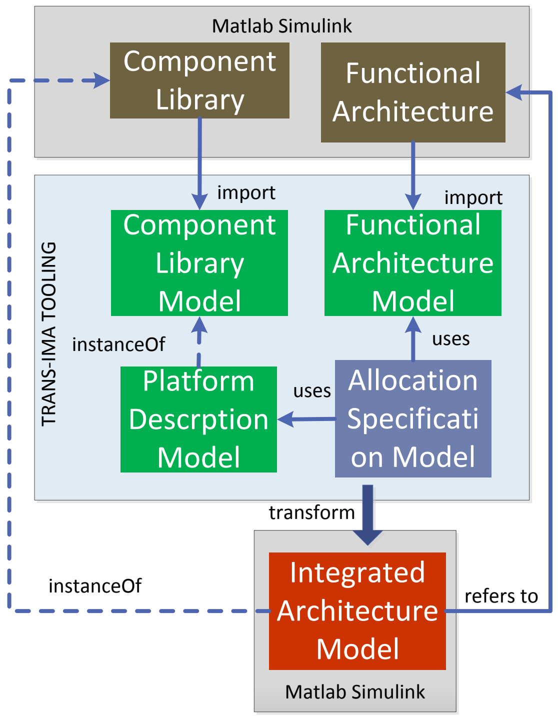

**Figure 6. Modeling architecture of Trans-IMA**

### *Input Matlab Simulink models*

The input models for the allocation process are the **Component Library** and the **Functional Architecture** defined in Matlab Simulink as a library and a system, respectively. Both of them are imported into the Trans-IMA framework carrying all relevant information required for the allocation process only, and does not include *any* other unecessary specification for example the internal behavior (e.g., defined by Stateflow).

### *Functional architecture model*

The goal of the *Functional Architecture Model* is to provide a cross-domain definition framework for capturing system functionalities and their corresponding information links. Its purpose is to help the system designer to focus on the high-level definition of the different functionalities and their interfacing without any platform/implementation specific details. Its most important building block is a *function* that represents a concept that later can be implemented either in software or hardware. Functions can contain other functions in any arbitrary depth, thus describing the required abstraction on the different levels. Additionally, it defines sensors and actuators for representing communication interfaces with the environment. A sample FAM is depicted in Figure 2

### *Platform description model and component library*

The goal of the *Platform Description Model* is twofold: (i) it describes the general hardware building blocks available (e.g., computational unit, router, chassis, power unit, etc) and also captures the different vendor specific versions of these general building blocks (e.g., Wind River PPMC74xx board, etc).

The model is always an instance of the *Component Library Model* (CLM) that is a Trans-IMA specific representation of the Component Library defined in Matlab Simulink. This special instance relation between the elements of the PDM and the CLM are captured using soft-links as provided by the EMF-IncQuery framework. More details about dynamic typing and traceability between models from different domains are discussed in [15]. Sample models for our case study are depicted in Figure 3Figure 4 andFigure 5.

### *Allocation specification model*

The *Allocation Specification Model* (ALS) contains all information that is defined during the Trans-IMA allocation process and is neither part of the FAM nor the PDM. This mainly includes: the mapping of the functions to the execution counterpart in the PDM, the information links to their corresponding communication paths, the routing tables for the communication paths and the software modeling components for the RTOSes (like, task, I/O Driver, etc. and partitions for IMA).

As an example from, the allocation of functions *AMS* and *Display* system from the motivating example (see the FAM in Figure 2) to partitions of *IMARTOS*es (see the PDM in Figure 3) is illustrated in Figure 7. The partitions are created automatically by the ALS editor (see more details in Section ***Allocation specification editor***) based on the selection of the user. Note that while the view shows the IMARTOS containing the partition which in turn encapsulates the function in order to help the user, the three elements are stored in separate models to allow fully flexible allocations. The two functions are connected with an information link and the view distinguishes links already allocated to paths by a different color (e.g., green).

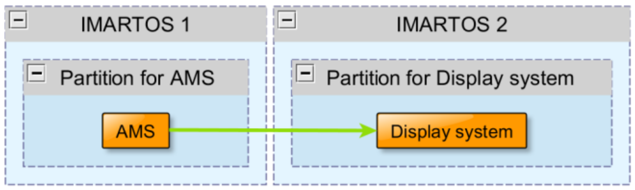

**Figure 7. Allocation specification, variant I.**

### *Integrated architecture model*

The final output of the development process is the *Integrated Architecture Model* (IAM). It is a Matlab Simulink model that is capable of simulating the allocated HW-SW system of the aircraft. It directly refers to the Simulink representation of the Functional Architecture Model and contains the instantiated Matlab Simulink elements of the Component Library as defined by the system architect in the Platform Description Model. Its ultimate goal is to be able to simulate the integrated system and through these simulations validate some of its dependability aspects.

## Overview of Trans-IMA tooling

A high-level overview of the internal architecture of the Trans-IMA framework is depicted in Figure 8. The different boxes represent different modules that encapsulate key functionalities of the overall system, while the red arrows represent the dataflow between these modules. These modules are explained in detail in the next section.

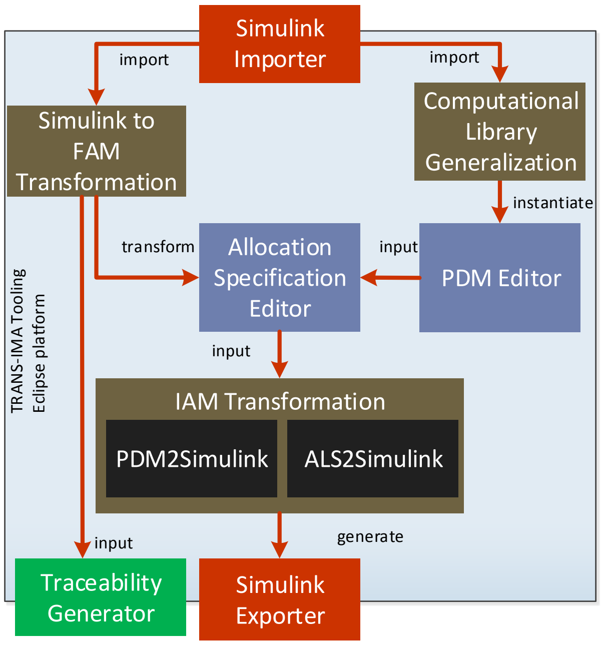

**Figure 8. Overview of the Trans-IMA framework**

### *Simulink exporter/importer*

This module handles the seamless integration of Matlab Simulink models and libraries to the Trans-IMA framework by providing a set of various import and export strategies.

### *Simulink to FAM transformation*

The Simulink to FAM model transformation step is responsible for automatically transforming the imported Simulink model into its corresponding FAM representation.

### *Computational library generalization*

The generalization step is used to import the different Computation library components (e.g., vendors' specific details, etc) to the Eclipse platform and generate their representing EMF models.

### *PDM editor*

Based on the imported Computation Library components the PDM Editor is used to design the platform description model. It is a graphical editor with multiple viewpoints for highlighting different segments (power, data etc.) of the underlying model.

### *Allocation specification editor*

The ALS Editor provides a high-level IDE for defining the allocation between the functional and hardware architectures specified in the FAM and the PDM. As its key aspects it supports (i) multiple viewpoints on the underlying system, (ii) automated communication channel routing in the hardware level and (iii) on-the-fly consistency validation using well-formedness constraints defined as EMF-IncQuery patterns.

### *IAM transformation*

The IAM Transformation is responsible for generating the simulation-ready IAM model for Matlab Simulink. It consists of two parts, where (i) the *PDM2Simulink* model transformation simply instantiates a Simulink representation of the Platform Description based on the defined PDM and its corresponding Computation Library elements and (ii) the *ALS2Simulink* transformation that is used to generate the Integrated Architecture Model based on the allocation defined in Allocation Specification Editor. Additionally, the transformation does not create a completely new Simulink model as it (re)uses the Simulink representations of the FAM (through the import process) and the generated PDM

### *Traceability generator*

To support the requirements imposed by DO-178C on traceability the Trans-IMA tooling provides a *Traceability Generator* module that generates traceability matrices for the complete process.

## Trans-IMA modules

### *Matlab-Simulink importer/exporter*

The main objective of the model importer/exporter is to allow manipulation of Simulink models within the Eclipse platform. Additionally, it provides direct traceability between the Matlab and Eclipse representations.

In order to support the Trans-IMA requirements our module provides the following key features: (i) access to a running Matlab instance, (ii) Simulink library and model reference support and (iii) multiple import modes and options.

#### Communicating with Matlab

Our solution for converting Simulink models is based on a direct connection to a **running Matlab instance**. To access and process the models we use command line functions and scripts. During communication, the client sends commands to a Java remote method invocation (RMI) based server component. The server evaluates them by using the **built-in Matlab Java interface**, and passes the return values back to the client after execution. In case of import, the result of this process is a serialized EMF representation of the Simulink model. Upon export, a Simulink system based on an EMF model is created.

As a unique feature, this integration also allows model initialization to be executed directly within Matlab. Furthermore, unlike many other external Simulink tools, that work directly on the persisted models, our solution uses the command line to process the models; therefore it is not affected by the various model format changes (e.g., mdl, slx).

#### Handling libraries and model references

In Simulink, models are built up from **blocks**, and blocks are collected in special models called **libraries**. Generally, models are created by copying blocks from libraries to models. Once a block is added to a model, a feature called a library link connects it to the original block. This keeps its internal structure synchronized with the block in the Simulink library as long as the link status is active.

Special blocks, called **model references** allow referencing complete, already existing models. This mechanism is similar to linking to library blocks, and needs to be supported as well.

These linking and referencing methods are respected by the importer/exporter. They do not only help the maintainability of the models, but also reduce the import/export time. For example, libraries imported only once can be referred multiple times.

To ease the allocation processes, customizable import traversal strategies are supported regarding library links and model references.

### Import traversal strategies

A traversal strategy consists of an **import mode** and arbitrary number of **import filters**. The Matlab importer/exporter module provides various import modes based on the aspects of hierarchical modeling of Matlab-Simulink:

- **Shallow**: only blocks within non-linked blocks are imported
- **Deep**: each block inside each subsystem is imported. Each referenced model is imported as an individual model with direct model referencing in the parent model
- **Flattening**: each model reference block is imported as though it was a subsystem
- **Referencing**: for blocks with active links, each source library is imported once as an individual model, but may be referenced multiple times

Additionally, filtering allows precise model element selection. This customizable option controls which hierarchy levels to import, based on the parameters of the blocks.

### Exporting models from EMF

The main purpose of the Simulink model export is to support the generation of complex IAM models. To achieve this, the following three features are supported: (i) to be able to reuse already existing models and libraries, (ii) automatic copying from libraries to models and (iii) creation of new systems.

## *Simulink to FAM transformation*

Imported Simulink models contain model elements such as **blocks, ports, signal connections** and properties. However, in the Trans-IMA approach the same imported model describes the functional architecture of the system, which represents **functions, sensors, actuators** and the data flow between them.

The challenge in creating a Simulink to FAM transformation is that an **automated abstraction** is required which derives an instance model of a DSL from an instance model described with general purpose Simulink.

The transformation identifies the type of elements based on **well-defined tags used in the subsystems** of the imported model. The Simulink model of the functional architecture is often embedded in an environment model that supplies inputs and checks outputs against expected results. Therefore, the transformation also has to **find the root blocks of the functional architecture**, which is done by using EMF-IncQuery to query subsystems with a given tag regardless of where they are in the containment hierarchy.

After creating the hierarchy of the functional architecture (root, intermediate and leaf functions), the transformation **identifies information links** represented by complex signal flows in Simulink. Outports of leaf functions may be connected to inports of other leaf functions through block hierarchy, bus creators and selectors, goto and from blocks. These **complex flows are compressed** by the transformation into direct links, while traceability information is also stored about which functional element represents which block, port or signal flow.

Finally, certain properties of the Simulink elements represent attributes of the functional elements, such as design assurance level (DAL) or a list of requirements that are related to them. The transformation **extracts the values of properties** and converts them into the proper data type (e.g. enumeration for DAL values instead of strings).

The transformation is defined as a set of **precondition-action rules**, where preconditions are model queries and actions are specified in Xtend.

## *Computational library generalization*

The platform description model specifies the hardware architecture of the system and contains elements representing routers, processing units, chassis, power supplies and different buses. These concepts are defined by the PDM metamodel and the instance models conform to it. However, a specific system architecture does not contain a generic router, it uses a specific router from a **library of available components** compiled from the offering of suppliers.

To support changes in such a computational library any time, this library is represented by a separate EMF model in Trans-IMA. However, since the aim of this approach is to simulate the functionality of the specified hardware in Simulink, the **elements of the library are also defined as Simulink models**.

Similarly to the FAM, the computational library is created as an output of an automated transformation. The elements of the library are subsystems in Simulink, and their tags and location in the Simulink library hierarchy are used to **identify the PDM concept that they specialize.** The traceability between the library component and the corresponding Simulink block is also stored and used in the IAM transformation step.

Since the computational library and the PDM models that use components from this library can be modified independently, it is important to handle cases where the component referred by a PDM element does not exist (e.g. it was deleted or an older version of the library is used, where it was not available yet). Storing simply a unique identifier of the component would make navigation, validation and editing cumbersome, therefore **soft links** managed by model queries **are used to create type references** between PDM element and component.

Apart from validating that the component selected for an element is indeed a specialization of the type of the element (e.g. a power supply must refer to a component that represents a specific power supply, not a router), further **constraints can be specified for components**. For example, a given router component may have a precise number of ports. These constraints are also validated on the PDM instance model.

## *PDM editor*

The specification of a complex platform description model involves several, often interconnected tasks, such as:

- Definition of the **physical containment hierarchy** (which modules are contained by a chassis, how many independent modules are there).
- Design of the **power connections** between producers and consumers.
- Specification of environment **sensors and actuators**.
- Description of communication ports and **data connections**.
- Specification of **operating systems and communication bridges** of processing units.

The PDM editor of the Trans-IMA approach supports all these tasks by extending the tree view based editor generated by EMF with **complex model editing commands** and different graph-based **views that focus on a given task**.

The set of available complex commands depends on the current selection. For example, if a data connection is possible between two selected elements, a command can create the required data ports and connections, which would otherwise require an error-prone sequence of atomic editing operations from the user.

Multiple graph-based views are created using the EMF-IncQuery viewer technology and visualized by yFiles. These **views highlight different segments** of the PDM model, for example the communication view shows data connections and the elements participating in them, but omits power supplies and power connections. The content and layout of the views are defined by annotations on declarative model queries, which make it **very easy to extend or modify** the views.

Finally, the live validation extension of EMF-IncQuery is also integrated with the PDM editor and it **alerts the designer immediately** if the constraints, specified by annotated model queries, are violated by the instance model.

### *Allocation specification editor*

When performing the allocation of functions to the platform, the editor has to provide visual information on:

- The **hierarchy of functions**, the information links between them and whether they are allocated or not.
- The **structure of the platform description** model, especially the data connection paths between processing units.
- The **details of the allocation** of both individual functions and the data connection paths selected for the information links.

While the allocation of a function is straightforward by simply selecting one of the available RTOSes, the **allocation of an information link is challenging**. First, only information links between elements that are already allocated can be selected. Next, based on where the elements are allocated, **all possible data connection paths are collected** from the PDM model. This process is helped by queries and uses a **specialized depth-first search** to identify all possible paths (even through routers and data buses). These paths are visualized in the editor in a separate yFiles view also driven by the EMF-IncQuery viewer extension.

The path selection is aided by **weight functions** (e.g. number of connections) which are evaluated on each path. When one of the paths is selected, **the allocation is stored in the ALS model**. This involves the configuration of software ports, IMA partitions, input-output drivers of communication bridges, and routing tables of routers.

As an example, consider that function *AMS* is allocated to the *IMARTOS* running on *ProcessingUnit 1* in *Chassis 1* (see Figure 3 and Figure 4) and function *Display system* is allocated to *Processing Unit 1* of *Chassis 2*. Figure 9 shows two possible communication paths that can be used for the information link between the two functions. Both alternatives pass through the communication bridge of their processing unit and the backplane data bus of the chassis. However, one path leads through AFDX Switch 1 using one set of I/O Modules, while the other passes AFDX Switch 2 through another set of I/O Modules. The ALS Editor can visualize any number of possible paths with overlapping parts correctly displayed without redundant elements.

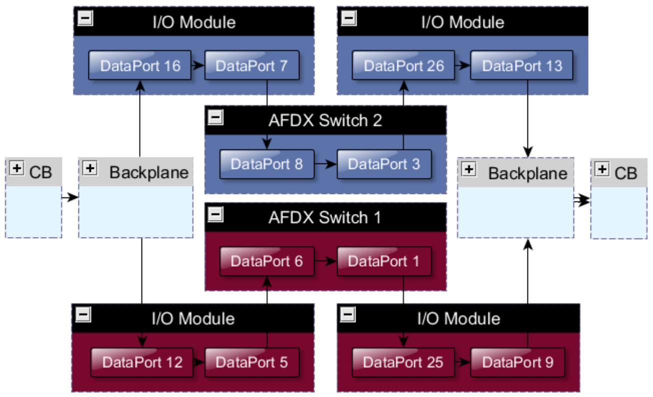

**Figure 9. Two path variants for the AMS allocation**

The same functional architecture may be used in different allocations; similarly, the same platform description may be used in several allocation specifications. Therefore, the **allocations specification editor must not modify the FAM and PDM models** in any way when creating the allocations. Additionally, the editor also **handles incomplete allocations**, when the PDM or FAM models changed after the allocation. To avoid inconsistency problems, the editor supports the deletion of allocations and properly removes configuration elements as needed.

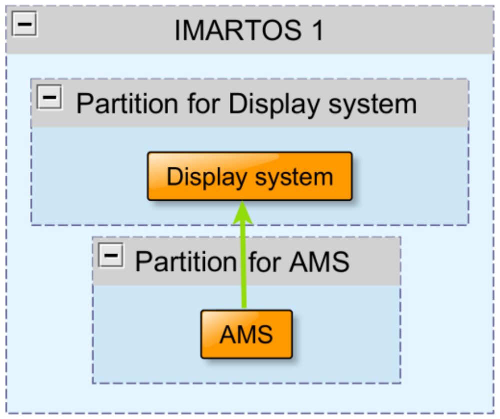

**Figure 10. Allocation specification, variant II.**

Figure 10 shows the functions AMS and Display system allocated to the same IMARTOS as a variant of the allocation described in Figure 7. Allocation specification, variant Although separate partitions are created for the two functions, the information link between them is not allocated manually, since no physical communication path is needed. Note that these different ALS models refer to the same FAM and PDM models and can describe design alternatives.

## *IAM transformation*

A complete allocation (i.e. when all functional elements and information links are allocated) provides all the information required to **synthetize an integrated architecture model** of the system that is **executable in Simulink**. The Simulink model is created in EMF and later exported with our Matlab-Simulink Exporter.

### PDM to Simulink transformation

The main block hierarchy of the IAM model is created based on the PDM by (i) traversing the model, (ii) copying blocks from the Computational Library for each element, (iii) preparing the power and data connections specified in the model.

The model is traversed by using model queries that **abstract the detailed containment hierarchy** of heterogeneous elements (e.g. chassis has slots with slotted elements and backplanes with buses).

Each PDM element has a component type from the Computational Library, and these components in turn represent blocks in a Simulink library. Model queries are used to **find the corresponding library block** for each element and copy it to the IAM model.

Finally, based on power and data connections in the PDM, the transformation **creates inports and outports** on the copied blocks and creates **signal flow connections** where specified. Note that the PDM ports are more complex than Simulink ports (e.g. it can represent fan-in, bidirectional ports); therefore often multiple Simulink ports are created for one PDM port.

### ALS to Simulink transformation

The IAM is completed by (i) inserting the blocks representing functions into the structure prepared by the PDM to Simulink transformation and (ii) configuring the signal flows for information links.

Functions in the FAM also correspond to library block in the input Simulink model. Based on the allocation, these library blocks are found by model queries and **inserted into tasks inside execution containers** (partitions for IMA or inside Federated RTOS).

The information links describe where the output of a function has to be routed, while the path through the PDM is stored in the ALS model. The transformation **creates the signal flow corresponding to the information link**, keeps track of how deep the signal is packaged into signal buses and creates availability signals for each segment that are used in the simulation to inject faults.

Finally, the transformation puts the complete IAM into a single library block which has the same interface (inports and outports) as the FAM, therefore it can be **inserted into the existing environment model for simulation**.

## *Traceability generator*

The Trans-IMA tooling uses multiple, interconnected models as illustrated on Figure 6. While the correspondence between model elements can be used to explore traceability through design artifacts, it is important to provide traceability information in a representation that is (i) stored separately from the artifacts themselves, (ii) remains in synch with changing models, (iii) supports queries and traceability specific views for navigating on traceability chains and (iv) provides export capabilities to document formats used in certification (e.g. traceability matrix).

### External traceability models

We created a traceability metamodel that defines concepts to represent domain models, their elements and traceability relationships between them, independent of the other Trans-IMA DSLs. This metamodel is extended in order to interconnect with the Simulink, FAM, PDM and IAM models, but the framework that handles traceability models deals only with the base metamodel. Therefore, the framework can be reused in different tools as well.

The external traceability models are created using annotated model queries that identify relationships between domain elements of different models. The queries and instance models are the inputs of the framework that interprets the queries, iterates through all relationships represented by query results and stores domain elements. It is possible to select different modes for this storage, including (i) direct reference, (ii) unique identifier or (iii) soft link derived feature.

**Synchronization of traceability models**

The domain models may change after the traceability model has been created. Since the models may be large, it is important to synchronize the contents of the traceability model without recreating existing parts. We use the event-driven rule engine of EMF-IncQuery to add, remove or update domain objects and relationships in the traceability models incrementally.

**Navigating on traceability chains**

One of the most important use cases of traceability is to navigate through a chain of links to follow the evolution of artifacts. For example, it is possible to select a Simulink block that represents a function, follow a traceability link to the FAM then another link to the function allocation in the ALS and on to the blocks in the final IAM model.

Once again, the visualization of traceability models and chains are supported through model queries over the traceability model and yFiles views driven by the viewer extension of EMF-IncQuery.

**Exporting traceability models**

While it is important to create and use traceability models inside the Trans-IMA tooling, there are cases when this traceability information is needed outside of the tool in a human readable (and often standard) format.

We have created a model-to-text transformation that uses model queries and Xtend to process a traceability model created in Trans-IMA and prepares a traceability matrix in HTML. This matrix can be explored, printed and used in certification to ensure that the allocation is correct.

## *Matlab-based simulation of IAM models*

Because the IAM is represented in Simulink, several analyses can be carried out using this model, specially leveraging some infrastructure already developed in previous projects at Embraer. The current section highlights a few of those analyses.

One of the difficult points of designing an IMA system is that single failures or spatially correlated failures may affect several systems at the same time, more so than with federated systems. For example, a memory or processor failure may bring down applications from two or three different systems at the same time. Certainly, depending on the criticality level of the functions, the system has to be designed in such a way that the effects of a failure like this are tolerable.

Due to the amount of failure modes and the different components, this analysis is very time consuming. One possible way to speed it up, especially in the early or middle design phases, when there are still several changes happening in the systems definitions, is to use simulation to ensure full coverage of all the failure cases.

As an example, suppose there is a CAS message whose absence (false negative) has catastrophic consequences. Therefore, it must be ensured that no single failure is able to cause this situation and that any combination of two failures has a probability rate smaller than $10^{-9}$ per flight hour to cause this event. A combinatorial search of all the pairs of failures may ensure these cases are covered or find all the possible cases faster than a traditional analysis would. This method can also be used to generate better fault-isolation rules for the systems. This usually has only minor safety impact but it is very important for reducing operational costs.

Another example were having a time-domain simulation model is useful is to analyze increase in crew workload due to failures. In this case, the IAM (plus the other models needed by the flight simulator) is loaded in the simulation and test runs are done with pilots.

Additionally, Simulink makes it easy to do several (but not all) timing-related simulations. Different physical platforms and different allocations may produce different time delays for some results. Some of these time delays may be very relevant for handling qualities. However, as a known disadvantage Simulink is not well suited for simulating the asynchronous aspects of the system.

The ultimate goal of the Trans-IMA approach is to provide significant speed up in the early simulation based analysis of the complete HW-SW system, where the tooling is aiming to ease the allocation process by the application of abstraction based model-driven engineering techniques.

## Related work

As the definition and configuration of avionics systems constitute a significant part of the complete aircraft development, research in the area is very active to provide better tooling and analysis both in academia and industry. The current section shortly highlights similar research directions on IMA and ARINC 653 design from the literature

One key area is to increase development effectiveness by using model-based techniques such as: early model-based validation and analysis using formal methods such as model checking and theorem proving [16], automated generation of application source code for ARINC 653 compatible RTOSs from high-level models [17], advanced IDE and validation support for the definition of ARINC-653 configuration artifacts such as Partition Operating System (POS), Module Operating System (MOS) and health monitoring tables [18], high-level architectural design of partition level application and their internal structure using state machines [17] and multi-aspect optimization of IMA systems taking into account simultaneously physical factors, software and hardware allocation and communication architecture in the optimization process [19].

Similar to our work, both [20] and [21] proposed a mainly simulation based early validation of IMA systems using the Cheddar and the SystemC simulation framework, respectively. However, both approaches focus on scheduling of partitions and their communication means, while in our case we simulate the behavior of the system in case of element level (e.g., partition, router, etc) errors and provide feedback on dependability characteristics.

Finally, our work in certain aspects is a direct continuation of the DIANA framework [22] developed in cooperation with Embraer, GMV, NLR and Thales in a European FP6 project. We applied similar contract based validation analysis for each design step and also followed the same abstraction based refinement and allocation concept to derive the ARINC 653 configuration artifacts for the mos, pos and health tables.

## Conclusion and future work

In the current paper, we introduced an abstraction based allocation approach for defining complex HW-SW avionics architectures using model-driven techniques to allow Simulink based analysis.

As an additional achievement the resulting Trans-IMA framework was built on state-of-the-art, open source technologies with Eclipse providing core functionalities and common user interface elements, Eclipse Modeling Framework (EMF) as the foundation for all domain models and EMF-IncQuery for the efficient, realization of the automated model queries and transformations. We plan to submit some parts of the framework to the PolarSys Working Group [23] to support the open source initiative with reusable tools.

As for the future we aim to enhance our framework to support: (i) the automatic generation of routing tables based on the allocation specification, (ii) the automatic calculation of the communication paths based on safety and bandwidth constraints and finally, (iii) automated instantiation and allocation of safety-critical redundant functions.

## References

[1] Eclipse Foundation, The Eclipse platform and IDE: http://www.eclipse.org.

[2] Moir, I., A. Seabridge, 2008, Aircraft systems: mechanical, electrical and avionics subsystems integration, John Wiley & Sons.

[3] The Object Management Group, Meta Object Facility (MOF) core specification version 2.0 http://www.omg.org/docs/formal/06-01-01.pdf.

[4] Eclipse Foundation, Eclipse Modeling Framework: http://www.eclipse.org/emf.

[5] ATLAS Transformation Language, http://www.eclipse.org/atl/.

[6] VIATRA2: VIsual Automated model TRAnsformations, http://wiki.eclipse.org/VIATRA2.

[7] GReAT: Graph Rewrite And Transformation http://www.escherinstitute.org/Plone/tools/suites/mic/great.

[8] EMF-IncQuery, http://www.eclipse.org/incquery/

[9] Bergmann, G., Z. Ujhelyi, I. Ráth, D. Varró, 2011, A Graph Query Language for EMF models, Theory and Practice of Model Transformations, Fourth International Conference, vol. 6707, pp. 167-182

[10] Ujhelyi, Z., G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, D. Varró, 2014, EMF-IncQuery: An Integrated Development Environment for Live Model Queries, Science of Computer Programming

[11] Xtend, http://www.eclipse.org/xtend

[12] yFiles for Java, http://www.yworks.com/yfiles

[13] International Society for Automotive Engineers, Architecture Analysis and Design Language, http://www.aadl.info.

[14] Modelica Association, The Modelica open-source declarative language, https://modelica.org/

[15] Hegedüs, Á., Á. Horváth, D. Varró, 2012, Query-driven soft-interconnections of EMF models, In Proceeding of the 15th International Conference on Model Driven Engineering Languages and System, Innsbruck, Austria, Springer.

[16] Miller, P. Steven., 2009, Bridging the Gap Between Model-Based Development and Model Checking, 2009, In Proc. of 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, York, UK, Springer, pp 443-453.

[17] Gamatie, A., C. Brunette, R. Delamare, T. Gautier, J.-P. Talpin, 2009. A Modeling Paradigm for Integrated Modular Avionics Design. In Software Engineering and Advanced Applications, pp 134–143.

[18] Choi, Eu-Teum, Ok-Kyoon Ha, Yong-Kee Jun, 2014, Configuration Tool for ARINC 653 Operating Systems, Vol. 9. No. 4. Int. Journal of Multimedia and Ubiquitous Engineering, SERSC, Australia.

[19] Annighofer, B.; E. Kleemann, F. Thielecke, 2013, Automated selection, sizing, and mapping of Integrated Modular Avionics Modules, Digital Avionics Systems Conference (DASC), IEEE/AIAA pp.2E2-1,2E2-15

[20] Lafaye, Michael, Marc Gatti, David Faura, Laurent Pautet, 2011. Model Driven Early Exploration of IMA Execution Platform. In Digital Avionics Systems Conference (DASC), IEEE/AIAA, pp. 7A2 -1 – 7A2 -11.

[21] Delange, Julien, Laurent Pautet, Alain Plantec, Mickael Kerboeuf, Frank Singhoff, Fabrice Kordon, 2009. Validate, Simulate, and Implement ARINC 653 Systems Using the AADL. In Proceedings of the ACM SIGAda annual international conference on Ada and related technologies, pp 31 – 44.

[22] Horváth, Á., D. Varró, T. Schoof, 2010, Mode-Driven Development of ARINC 653 Configuration Tables, In Digital Avionics Systems Conference (DASC) , IEEE/AIAA, pp 5.A.5-1 – 5.A.5-115

[23] PolarSys, Open Source Tools for Embedded Systems, http://polarsys.org/

## Acknowledgements

This work was partially supported by the CONCERTO (ART-2012-333053).

## Email addresses

Ákos Horváth: ahorvath@mit.bme.hu

Rodrigo R. Starr: rodrigo.starr@embraer.com.br

Samoel Mirachi
s:amoel.mirachi@embraer.com.br