MODEL-DRIVEN DEVELOPMENT OF ARINC 653 CONFIGURATION TABLES

Ákos Horváth, Dániel Varró, Budapest University of Technology and Economics
Department of Measurement and Information, Budapest, Hungary
Tobias Schoofs, GMV, Lisbon, Portugal

Abstract

Model-driven development (MDD) has become a key technique in systems and software engineering, including the aeronautic domain. It facilitates on systematic use of models from a very early phase of the design process and through various model transformation steps (semi-)automatically generates source code and documentation. However, on one hand, the use of model-driven approaches for the development of configuration data is not as widely used as for source code synthesis. On the other hand, we believe that, particular systems that make heavy use of configuration tables like the ARINC 653 standard can benefit from model-driven design by (i) automating error-prone configuration file editing and (ii) using model based validation for early error detection.

In this paper, we will present the results of the European project DIANA that investigated the use of MDD in the context of Integrated Modular Avionics (IMA) and the ARINC 653 standard. In the scope of the project, a tool chain was implemented that generates ARINC 653 configuration tables from high-level architecture models. The tool chain was integrated with different target systems (VxWorks 653, SIMA) and evaluated during case studies with real-world and real-sized avionics applications.

Introduction

The ARINC 653 standard [1] has taken a leading role within the aeronautical industry in the development of safety-critical systems based on the Integrated Modular Avionics (IMA) concept. One of the main promises of IMA is cost saving in reduced development, integration and verification and validation effort.

In case of ARINC 653 compliant platforms many deployment and implementation details are defined in the configuration tables. Typically, these configurations are hand defined by the system architect with limited tool support that only ease (i) the manipulation of its XML representation, (ii) their validation to the ARINC 653 schema definition and some consistency checks.

Unfortunately, despite the inherent complexity of ARINC 653 configurations, current tools supporting configuration design offer very low-level support directly on the XML representation level. However, existing tools lack support for (1) capturing the development process for configurations, (2) validating design constraints for configurations on-the-fly, (3) recording explicitly the critical design decisions made by the system architect, and (4) providing traceability between high-level requirements and the configuration tables, which require hand-crafted traceability lists. As a result, verification of configuration tables is a tedious activity.

Model-driven development (MDD) has become a key technique in systems and software engineering. It facilitates on systematic use of models from a very early phase of the design process. Based on high-level modeling standards (like UML , SysML [2] or AADL [3]), traditional MDD separates business and application logic from underlying platform technology by using platform independent models (PIM) to capture the system requirements, and platform specific models (PSM) to specify the target system on the implementation platforms (ADA, Java, C++). PSM may refer to models or to platform-specific artifacts like source code and configuration elements; the latter are automatically generated from PIM and PSMs, respectively, using automatic model transformations.

However, as MDD is attracting increasing attention in safety-critical system development [4], the original approach needs to be adapted to be in-line with the rigid certification requirements (e.g., DO-178B [5]) imposed by authorities.

In the paper, we present a framework for systematically designing standard ARINC 653 configuration tables with additional support for configuring (i) the Wind River VxWorks 653 Safety Critical RTOS [6] and (ii) the GMV SIMA ARINC 653 simulation platform [7]. Additionally, parallel to the development process our approach generates end-to-end traceability information to support certification Our toolkit is implemented in the Eclipse framework, and it is built upon the principles of Model-Driven Development (MDD).

The framework was developed in the context of the DIANA [8] project financed by the European Commission through the Sixth Framework Programme in close collaboration with leading avionics experts and airframers including GMV, AleniaSia, Atego, Dassault, Embraer, NLR, THALES, and academic partners of TU Budapest and Karlsruhe Institute of Technology.

Outline

In order to introduce our approach we (i) outline the basics of model-driven development for safety critical system, (ii) presents our intermediate models and target platforms (iii) give a motivating air conditioning case study, (iv) introduce our PIM-PSM mapping approach, (v) highlight a contract based V&V approach and finally (vi) conclude the work.

Model-Driven Development for Safety Critical Systems

Models are prime artifacts of engineering. In system development, they have played an important role as a way to capture real world notions as well as abstract constructs. In fact, system architects have been using models and modeling techniques long before model-driven development emerged as a trend, e.g. in the form of entity-relationship diagrams, graph-like data structures, abstract syntax trees etc. However, the term *Model-Driven Development* (MDD) implies that models play a central role that encompasses the entire system development lifecycle, starting from requirement analysis, system design, implementation, to verification and even maintenance.

Model-driven development aims to increase the efficiency and productivity of the software development process by introducing precise engineering practices based on formal modeling techniques. By this approach, design intelligence is applied to capture all relevant information in the form of abstract models. First, these models can be used for *documentation* purposes to store well-structured information about the system-under-design. Moreover, models can also be used for *generative development*, where *target design artifacts* (source code, configuration tables, test cases, textual documentation, etc.) is (semi-)automatically derived by tools. Finally, models can also be used for *early validation*, where important properties of the products (such as reliability, performance, robustness, security, complexity) can be evaluated before actual implementation begins. All of these techniques aim at *reducing costs and risks*.

MDD emphasizes the clear distinction between *Platform Independent Models* (PIM) and *Platform Specific Models* (PSM), thus, software development in MDD is envisioned as a three-step process.

First, the **Platform Independent Model (PIM)** is designed. The main purpose of this model is to capture the underlying business logic without specific implementation details and, this way, help portability to other *target platforms* (e.g., a prototyping platform, using Java, for instance; the final target platform, ARINC 653 for the aeronautical world or AUTOSAR [9] for the automobile domain based on ADA or C).

The second step is to generate a **Platform Specific Model (PSM)**, which contains additional models, and represents an implementation of the system under design which can run on the target platform. The transition between PIM and PSM (**PIM-PSM mapping**), should typically be facilitated using automated model transformation technology.

Finally, **software artifacts** (e.g., configuration files, source code, documentation, etc.) **are generated** from the Platform Specific Model for the target platform. Again, code generation should be as extensive as possible, in order to minimize the amount of necessarily slow and error-prone manual coding. This, in turn, requires PSMs that are expressive enough, not only from a static, but also from a dynamic point of view of the system, to produce all of the application artifacts.

Enabling technologies

MDD 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) [10] from OMG and the *Eclipse Modeling Framework* (EMF) [11] (a subset of MOF).

Model transformations (MT) are the backbone of the MDD concept. Primarily, model transformations are responsible for the PIM-to-PSM transformations. 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) [12], the *VIATRA2* (VIsual Automated model TRAnsformations) system [13] and the GReAT (Graph Rewrite And Transformation) framework [14].

Challenges in MDD for Safety Critical Systems

In order to support the specific needs of the safety-critical development processes, we followed the guidelines introduced in the EU-FP7 INDEXYS [15] project for the definition of a MDD means for embedded systems. Based on these guidelines the main challenges of MDD for safety-critical systems are the following (depicted in Figure 1.):

V&V activities need to **be tightly integrated** [16] into the development process to provide early feedback on requirements, specification, design and implementation. This requires a continuous verification activity from early specification through design to development. On top of that it has to be **in-line** with rigid **certification requirements** (e.g., DO-178B) imposed by authorities like FAA or EASA.

The **PIM-PSM mapping process** [17] needs to support both *automatic* and *user-driven* design steps as many critical design decisions are taken during the mapping process and cannot be fully automated. Furthermore, these decisions need to be recorded for traceability related certification requirements.

PSM needs to support the **different viewpoints of the system** [17] with a systematic separation of system level aspects (e.g., functionality, dependability, security) and a strong separation between *architectural* and *behavioral* aspects. This allows to use appropriate COTS or proprietary tools for the generation of textual artifacts.

Finally, PSM needs to support **synthesis** not only for source code but also system **configuration**, **certification** and **documentation** artifacts.

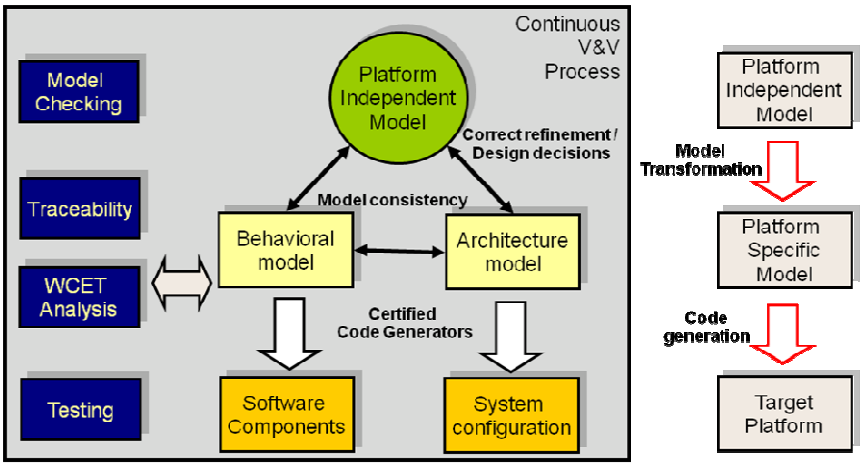

Figure 1: MDD for Safety Critical System Development

Modeling Architecture of the DIANA approach

Within the DIANA project one of the main goals was to create an MDD based tool chain for the analysis and generation of ARINC 653 real-time operating system (RTOS) configuration files from high-level specifications captured as *platform independent models*. However, transforming these high-level models into *ARINC 653 RTOS-specific* configuration artifacts is a complex task, which needs to bridge a large abstraction gap by integrating various tools. Moreover, critical design decisions are also made during this mapping process. For this reason, we used *intermediate domain specific models* to subdivide the process into well-defined steps. The overview of the model architecture is depicted in Figure 2

Platform Independent Models:

In our approach the aim of the high-level Platform Independent Model (PIM) is to capture the high-level architectural view of the system along with the definition of the underlying implementation platform, while the Platform Specific Model (PSM) focuses on the communication details and service descriptions. All our models are defined as separate EMF models.

In order to support already existing modeling tools and languages (e.g., Matlab Simulink model, SysML, etc.) we use a common architecture description language called *Platform Independent Architecture Description Language* (PIADL) for the description of architectural details by extracting relevant information from common off-the-shelf modeling tools. As for capturing the underlying platform (in our case ARINC 653) we use a *Platform Description* model (PD) capable of describing common resource elements. Functional requirements are also incorporated into the PIADL along with the Platform Description.

- *PIADL* aims to provide a platform independent architectural-level description of event-based and time-triggered embedded systems using message communication between applications.
- The *Platform Description* (model) describes the resource building blocks, which are available to build a system. This mainly includes ARINC 653 based elements such as modules, partitions, communication channels, etc.
- In the context of the DIANA project we supported *MATLAB Simulink* as a source COTS language.

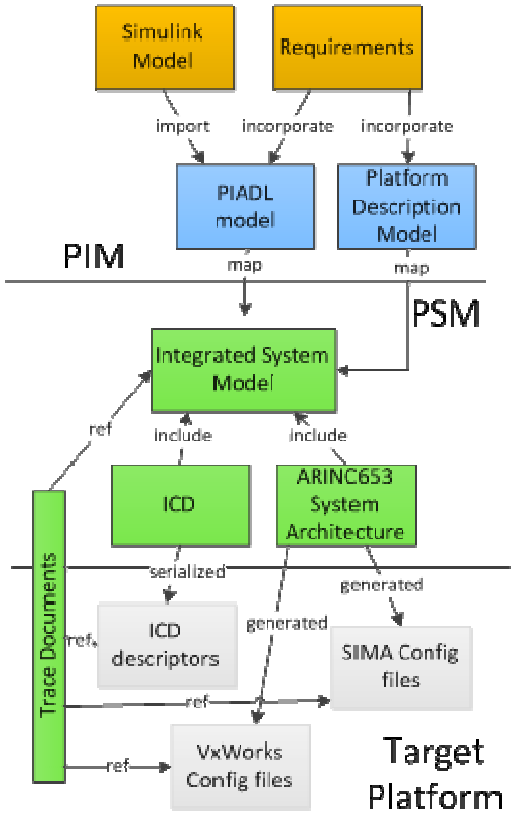

Figure 2: Modeling Architecture of the DIANA approach

Platform Specific Models

PSMs are encapsulated in the so-called *Integrated System Model* that contains all relevant low-level details of the modeled system. Essentially, it is based on ARINC 653 and consists of the following sub-documents:

- The *Interface Control Document* (ICD) is used to describe data structures and low-

level data representation of systems and interfaces to ease integration of the described element with other parts of the system. It supports both high-level (logical) and low-level (physical) descriptions and was designed to be compatible with the ARINC 653 and ARINC 825 data and application interface descriptions. Its descriptors are simple XML files containing the serialized form of the model describing the defined data structures.

- The ARINC 653 *System Architecture* model describes the relations among all elements related to the system. More precisely the model focuses on the (i) details of the communication channels between applications, partitions and modules, and (iii) the detailed allocation of the applications to partitions.

In order to support **traceability,** a trace element is saved in the *Trace documents* for all model elements of the PSM created during the mapping process. Such an element saves all PIM model segments that were used for the creation of a PSM model element.

Target Platforms

During the DIANA project, two OS target platforms were used: Wind River's VxWorks 653 real-time operating system and GMV's ARINC 653 simulator SIMA running on Linux [18]. The following sections introduce these platforms briefly and describe peculiarities of their configuration tool chains.

Wind River VxWorks 653 RTOS

VxWorks 653 is Wind River's platform for safety-critical applications certifiable according to DO-178B [19]. It is an IMA operating system with proven compliance to ARINC 653 [20][21].

VxWorks 653 implements IMA by means of virtualization technology [6]. There is a hypervisor monitoring and controling a set of guests. Each guest uses its own local executive, the Partition Operating System (POS). Several types of POS are supported by the platform, such as the ARINC 653 APEX, the classic VxWorks RTOS or a general purpose OS like Linux. Note that there is only one code instance per POS physically present in the system that is linked to the virtual address space of the partitions that actually use this particular POS.

The hypervisor is called the Module Operating System (MOS). It implements time- and space partitioning, the ARINC 653 inter-partition communication channels and the Health-Monitoring system. The MOS is the only component that runs in privileged mode. Guest systems run in user space and are not allowed to execute privileged instructions that may impact the proper function of the system.

Figure 3 (based on [6]) illustrates the architecture:

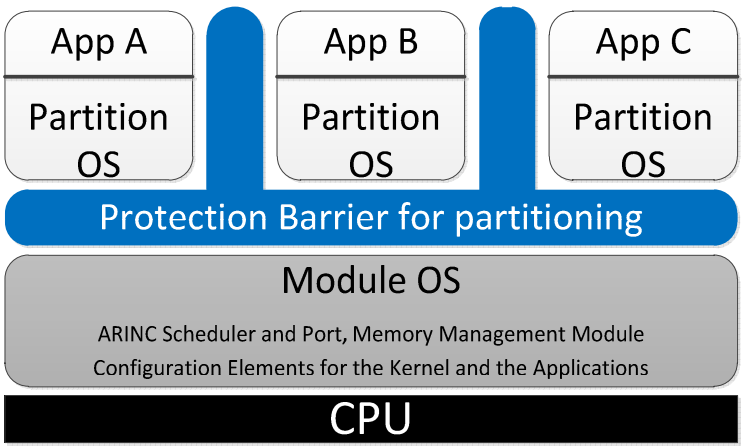

Figure 3: VxWorks 653 Architecture

Note that the components of the system are not linked together to one image; instead individual binaries are created for the MOS, for the POSes and for the applications. The boot loader is responsible to locate the different components on the boot medium and to load them into memory according to a configuration derived from system configurations.

The configuration the system integrator has to provide in order to link, load and execute the system follows the VxWorks component structure [22]. There is a configuration file for the MOS that defines fundamental architecture-related settings, such as processor frequency, page size and virtual and physical memory; there are configuration files for the POSes, defining their memory layout and how they are loaded into memory; there are configuration files for the applications, defining memory sizes and ports; there are Health Monitor tables that define the health monitoring on partition and module level; there is, finally, a configuration for the module bringing the single configuration files together and adding time partitioning-related information.

This configuration is different from the configuration defined by supplement 2 of ARINC 653. However, the next supplement will present a new approach: The standard will define a set of data types that must be used for an ARINC 653-compliant configuration, but will not impose a schema that describes how the elements must be related. The schema is left to implementations.

This approach of the ARINC 653 subcommittee is just a consequence of the fact that today's operating systems do not comply with the standard schema. Configuration is tightly coupled with the OS architecture and, as such, is difficult to standardize. For the task of generating vendor-independent valid configurations, this is probably not good news. Tools are needed that deal with the heterogeneity of configurations.

GMV SIMA simulator

Simulated Integrated Modular Avionics (SIMA) is an execution environment, providing the ARINC 653 Application Programming Interface (API) and robust partitioning to operating systems that do not support these features by themselves [7]. SIMA is designed to run on all POSIX-compliant OSes and optimised for the Native POSIX Thread Library (NPTL), available on Linux since kernel version 2.6.

In SIMA, ARINC 653 partitions are mapped to POSIX processes, and ARINC 653 processes are mapped to POSIX threads. Each SIMA application is, hence, linked to a single POSIX program, containing user code and data, the APEX code and data and, finally, the platform execution environment, i.e. the NPTL for Linux.

The Module Operating System (MOS), controlling the different POSIX processes, belonging to the same simulated module, is likewise linked to one POSIX process. The following picture illustrates this design:

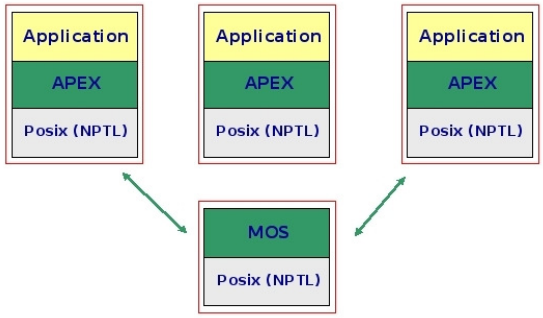

Figure 4: SIMA Architecture

The APEX services are implemented by a static library, called POS. The POS implements the APEX process scheduler on top of the POSIX FIFO scheduler (*sched_fifo*). POSIX features are encapsulated within a portability layer; this way main parts of the APEX code do not rely directly on POSIX, but on scheduling policies implemented by the POS itself. The advantage of this approach is enhanced portability - there is even an implementation of the SIMA POS, running on bare hardware - and the fact that scheduler features that introduce subtle differences between different POSIX implementations are handled in the portability layer and hidden from the APEX implementation.

The MOS implements the APEX partition scheduler. To be able to suspend and resume partitions, commands are exchanged with the POS in the partitions using signals and shared memory segments. Obviously, this approach does not answer safety and security threats, caused by random errors in the partitioned code. The POS has to respond

correctly to given commands which may not be true in the case where faulty or malicious application code corrupts the state of the POS. In fact, the MOS does only simulate the behaviour of an ARINC 653 compliant OS on top of non-safety aware systems like standard Linux.

Since SIMA main purpose is simulation, it aims at full conformity with the standard. The SIMA configuration is therefore strictly compliant to the schema defined in today's ARINC 653 part 1 and 2. Additional information that is needed by the system is added by means of a separate configuration file. This file defines the mapping of certain elements of the ARINC 653 configuration to the Linux OS; APEX ports, for instance, can be mapped to UDP ports.

Case Study: Air Conditioning

In order to introduce our approach, let us assume a generic air conditioning system installed on an airplane.

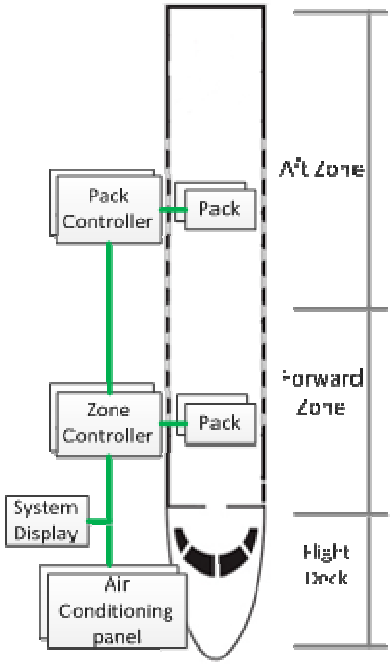

Figure 5: Overview of the Air Conditioning Case Study

Its task is to regulate the temperature and pressure in the aircraft. This is done in the following way. The air conditioning *pack* is regulated by the *pack controller* to supply the mixing unit with a sufficient flow of cool fresh air. This air is supplied to arbitrary number of *zones* (in Figure 5 we depicted two zones *Aft.* and *Forward*). In order to regulate the temperature of this airflow, the *zone controller* regulates the amount of hot air added to the flow of cool air, which is set on the *air conditioning panel* and monitored on the *system display*. Additionally, as air-conditioning is a critical task all systems have a redundant equivalent for better reliability.

An overview of the air conditioning system is depicted in Figure 5. Throughout the paper we will use this case study as our running example. It is a simplified version of the NLR demonstrator in the DIANA project.

Steps of the PIM-PSM Mapping Process

In order to introduce our PIM-PSM mapping concept we first, highlight the steps of a general PIM-PSM mapping process, then go into details about our concrete implementation.

A general PIM-PSM Mapping Process for Safety Critical Systems

A general PIM-PSM mapping process, in the safety-critical system design domain, consist at least the following steps (see in Figure 6):

1. *Define / Derive the platform-independent system model (PIM)***.** The architecture-level integrated system design starts by specifying a set of applications attributed with properties extracted from the system requirements (functional and non-functional) and high-level initial system models captured in SysML [2], AADL [3], etc.. These properties are captured in the **PIM model**.

2. *Define / Derive the Platform Description Model (PD).* The **PD model** describes all the details (CPU, latency, bandwidth, etc) of the underlying hardware platform including cabinet specification and hardware resource descriptions

3. *Define / Derive Platform Interface (PI).* This model describes the high-level middleware services offered by the underlying platform.

4. *Extract design constraints (performance, dependability, security, etc.).* The PIM and PD models should also include **design constraints,** which have to be satisfied by valid PSM models derived as a result of a PIM-PSM mapping. These constraints are extracted from the functional and non-functional System requirements (e.g., modular redundancy).

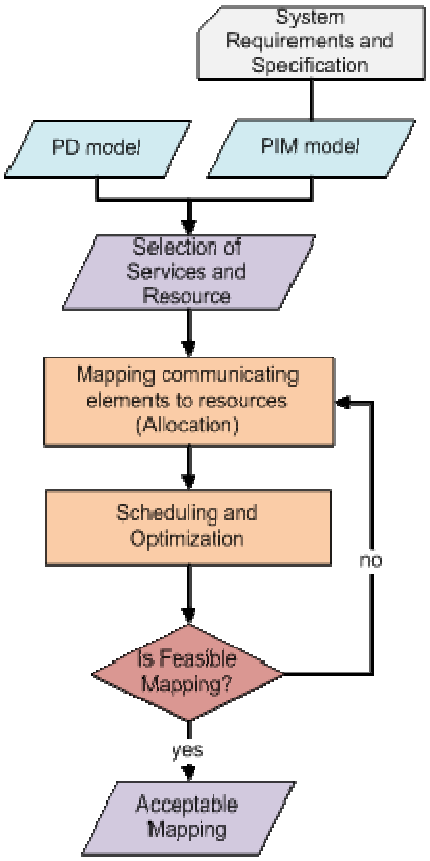

Figure 6: A Generic Mapping Process

5. *Define variability points / design choices.* As there is more than one possible mapping of a PIM to a target PSM, the PIM-PSM mapping should offer **variability points** to explicitly capture design choices. These variability points can be subject to future optimization steps.

6. *Resource allocation.* As the core phase, the system architect **assigns application types to resources** (called **resource allocation**), which provide general rules / guidelines for the PIM-PSM mapping. From these high-level guidelines, the actual mapping instances (i.e. mapping of application instances to actual resources) could be partially automated to obtain the **PSM model**. A valid PIM-PSM mapping should fulfill all design constraints (e.g. should not exceed HW limits like available memory).

7. *Scheduling and Optimization.* In addition, further **scheduling and optimization** steps can be carried out after resource allocation, which is out of scope for the current paper. For further reference see in FRESCOR [23]

8. *Evaluate the quality of the mapping.* The quality of the mapping can be evaluated against all functional and non-functional requirements, and certification means.

The DIANA Approach

We support the system architect by subdividing the *PIM-PSM mapping process* into well-defined design steps and by the precise definition of the interactions and interfaces at each step. Individual design steps are then organized into a complex *workflow* [24], which is closely aligned with the designated development process followed by the system builder (system integrator, function provider and platform provider.). In order to assist the system architect, our framework guarantees that a certain design step can only be started if all prerequisite steps are successfully completed. Our framework is easily customizable to incorporate additional design steps, if required.

The high level workflow of the PIM-PSM mapping process as used in the DIANA project is depicted in Figure 7. The process consists of 22 steps organized into five groups.

To illustrate some technicalities of our approach we use the simplified Simulink model (depicted in Figure 8) as the starting PIM model of the air conditioning case study.

Application Group

The group consists of steps to define the resource requirements of the *applications* and *partitions* used in a module and create a viable mapping that is compatible with the available resources and dependability requirements.

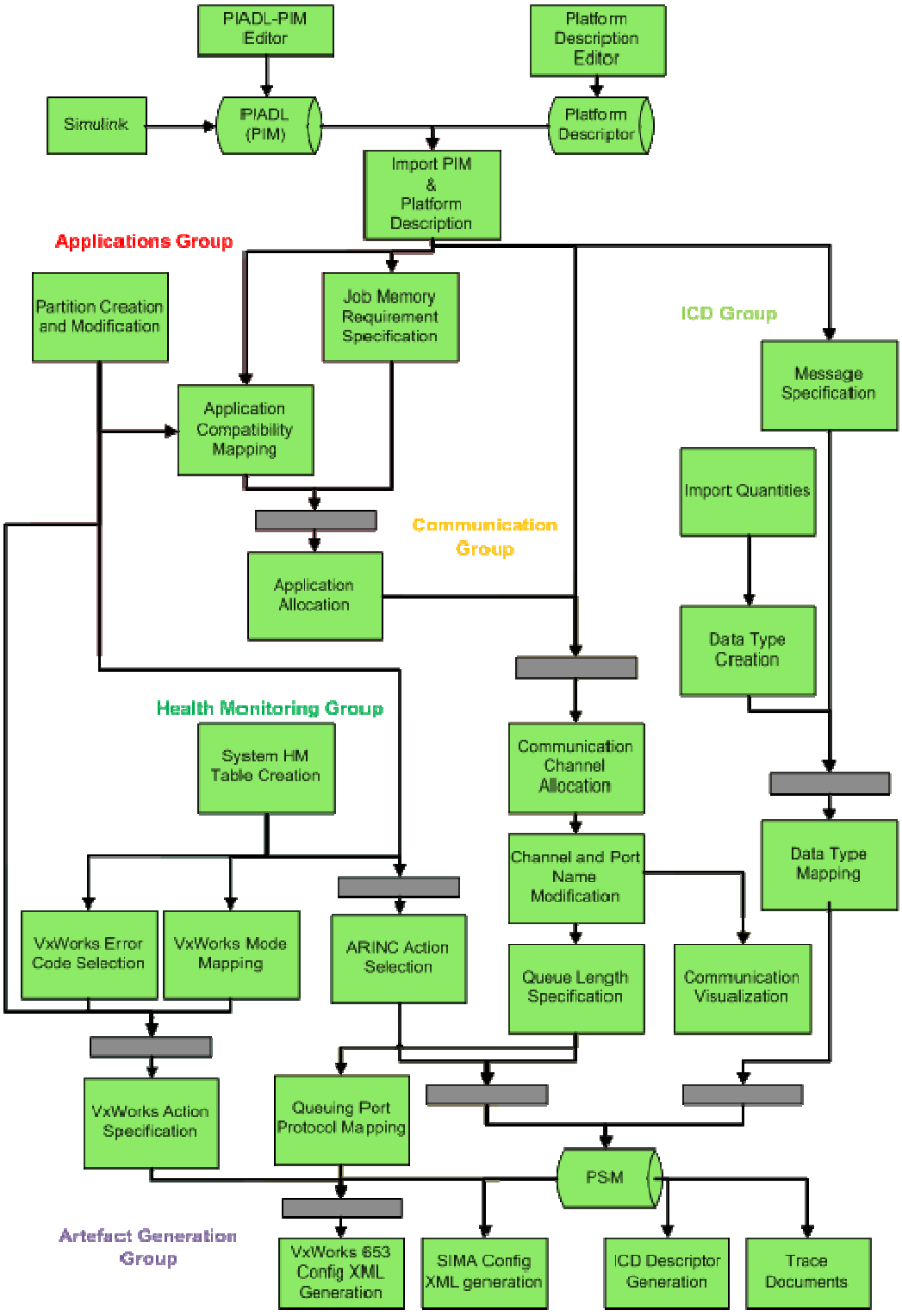

Figure 7: Workflow of the DIANA PIM-PSM mapping process

First, the PIADL and the PD models are imported into the framework. This step also resolves certain dependability attributes defined in the PIADL like redundancy degree of applications and messages (e.g., triple or double modular redundancy etc.).

As the platform description does not include all information needed for the allocation process and configuration generation, the system architect needs to (i) define the memory requirements and compatibility mapping of the applications and (ii) new partitions or modify existing ones and define their memory requirements.

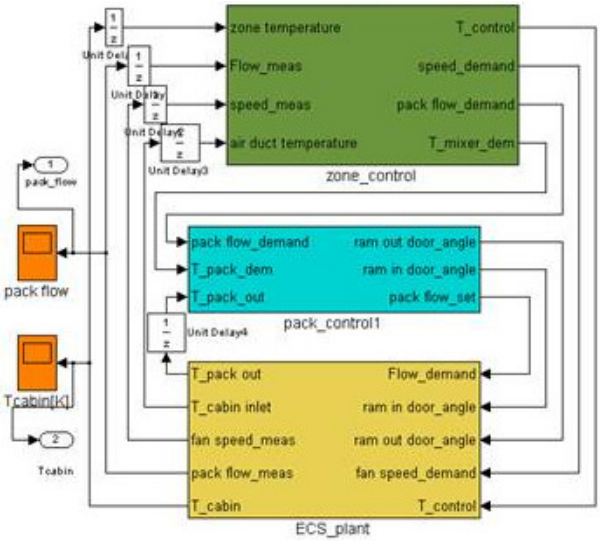

Figure 8: PIM model of the Air Conditioning Case Study[1]

To demonstrate how these steps are captured on model level, Figure 9 illustrates the low level model elements created for a partition (*partitions creation step*). Model Elements in yellow and dashed lines are newly created, while elements in green (and solid references) are already existing in the model. The tags <<*Integration*>> and <<*PD*>> represent the package of the model element. Partitions are defined/stored in the Platform Description model with separate model elements describing their corresponding memory requirements capturing the size, access (type) and type attributes. PSMRoot is the root element of the integration model and it holds references between the elements of the PIADL, PD and the PSM models.

For easier readability (i) attribute types are excluded from the figure and (ii) references and association are depicted by simple lines.

As the final step in the group, all allocations of applications-to-partitions conform to the defined constraints and requirements are computed. This way the system architect can select a valid allocation and (if required) can take into account additional non-functional requirements.

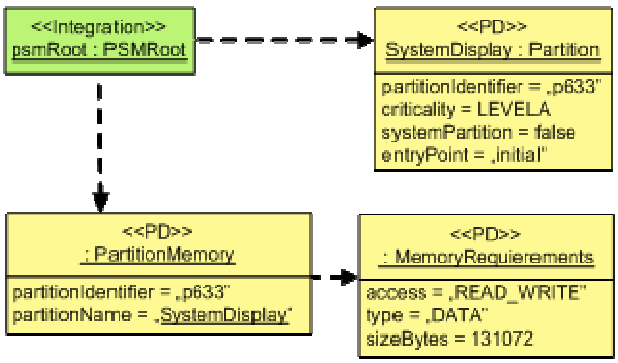

Figure 9: Partition Creation

The allocation problem is solved as a constraint satisfaction problem.

Communication Group

The group involves steps in the PIM-PSM Mapping editor that carry out the allocation of inter-partition communication channels and the specification of ports residing on each end of these channels.

The allocation is based on the architecture defined in the PIADL model (derived from a Simulink model), the selected application to partition mapping and the redundancy requirements of the applications. Based on this information the allocation algorithm creates the required ARINC 653 ports and connects them.

Additionally, the system architect needs to define the ARINC 653 specific parts like the queue length and the VxWorks specific queuing protocol to be able to generate the configuration files.

Figure 10 depicts a simple example how the allocated channels are visualised. In this case *Data Monitoring* application allocated over the *I/O Processing* partitions uses the *Temp*. channel to send the temperature value to the *Refresh GUI* application.

[1] © 2010 NLR

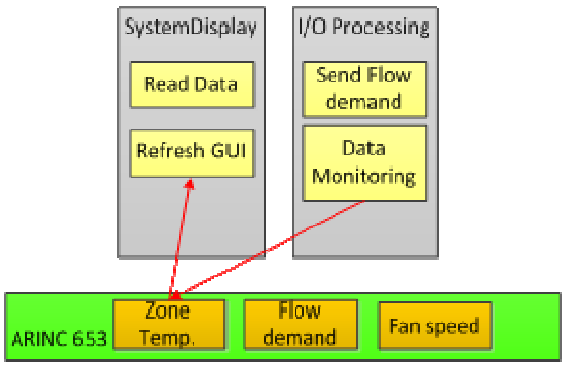

Figure 10: Allocated Communication Channels

Health Monitoring Group

The group consists of steps to define the Health Monitoring tables for module, partition and application level along with the different error entities and actions to be carried out.

All these definitions are done by the system architect by hand. The framework gives support for early validation (e.g., naming conventions, required action definitions etc.) based on the specification of the different tables and the system-specific requirements for health monitoring tables..

The defined tables are saved in the Platform Description model with the appropriate references from the PSMRoot of the Integration model.

ICD Group

In this group steps related to the description of interfaces and messages provided and required by applications. These are user driven mapping steps, where PIM types and messages, are refined with platform specific information like encoding, default value, etc.

Figure 11 describes how the *Temp* PIM type is refined into the *Int1_100* PSM representing an integer value with a domain of 1-100. The *Int1_100* type is based on the predefined *16 bit unsigned integer type* from the ICD with additional constraints over its domain. Based on these PSM types, complex messages are defined following a similar way, where the ICD provides the basic structures like arrays, buffer, etc.

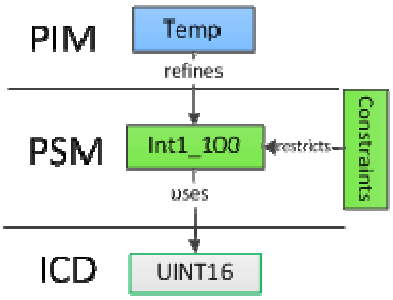

Figure 11: Definition of the Temp value in ICD

Artifact Generation Group

Finally, when the prerequisite steps for a certain code generator is finished the actual textual representation is synthesized by separate dedicated code generators.

In our case the ICD generator simple serializes the model into its XML representation using the built in support of the Eclipse Modeling Framework. As for the other artifacts we hand-coded the generators in java to derive the required formats defined by the two platforms.

The communication architecture, depicted in Figure 8, is mapped to ARINC 653 ports through the defined mapping process and then automatically generated both the ARINC 653 and the VxWorks 653 specific *module* and *ApplicationDescription* XML configuration tables. A fragment of the generated configuration tables capturing the communication channel depicted in Figure 10 is captured in Figure 12.

```
<!-- Arinc 653 Module -->
<Channel Id="9">
 <Source PartitionNameRef="IOProcessing"
  PortNameRef="IO_SD_2_S"/>
 <Destination PartitionNameRef="SystemDisplay"
  PortNameRef="IO_SD_2_D"/>
</Channel>
<!--VxWorks ApplicationDescription-->
<SamplingPort Direction="DESTINATION"
MessageSize="16" Name="IO_SD_2_D"
RefreshRate="0.2"/>
```

Figure 12: Example ARINC 653 Module and VxWorks Application Description configuration

Traceability

Additionally, as an essential requirement of DO-178B certification, *continuous traceability* has been carried out from the high-level requirements to the deployed applications (depicted in Figure 13).

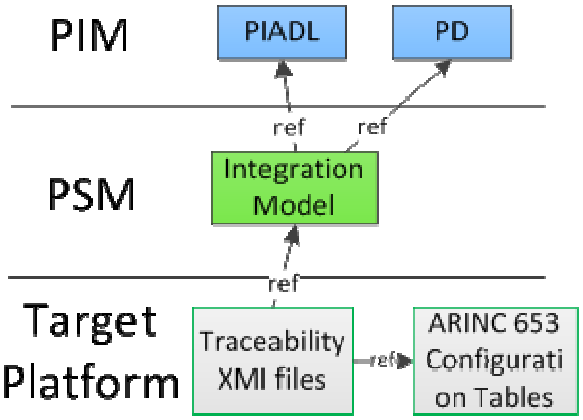

Figure 13: Traceability between models and configuration artifacts

In case of the design phase we used (i) inter-model traceability based on the *Integration Model* that keeps track of all manipulations done during the PIM-PSM mapping process and (ii) model-to-configuration traceability with XMI files connecting generated configuration elements to their corresponding model entities. This allowed end-to-end traceability from the PIADL model to the generated configuration tables. It is important to mention that the current implementation requires an explicit definition of traceability elements between the various models. However, currently we are investigating special *live model transformations* [25] to give support for automatic generation of traceability elements without explicit definition.

Verification and Validation Support

Keeping the design and verification aspects tightly synchronized, enables early validation as close as possible to the corresponding model/code development time, thus providing better feedback and error detection. To support early validation of modeling artifacts during our development process we used contracts to guard each steps.

Contracts

During a development process certain steps require external COTS tools (e.g., Matlab, SAL, etc.) or user interaction to perform their task. In order to guarantee that the result of these steps is *acceptable* and the process can *continue*, the definition of contracts [26] is a well-known paradigm.

The idea is to *guard* both the input and output of a step by specific constraints. Thus, a contract is composed of a *precondition* and a *postcondition*. In our interpretation a precondition defines constraints that need to be fulfilled by the input of the step in order to allow its execution, while the postcondition guarantees that the process can continue only if its constraints are satisfied by the output.

In our approach we used *graph patterns (GP)* [27] to capture such contracts. GPs are frequently considered as the atomic units of model transformations. They represent conditions (or constraints) that have to be satisfied by a part of the underlying model. Based on these contracts we investigated two promising approaches to support early validation and verification.

On-The-Fly Evaluation of Contracts

One of the main advantage using contracts to specify constraints on the input and output (model) of each step in the development process is that it allows fine grain (step level) validation of model changes throughout the whole workflow. However, graph patterns can express *complex model constraints* containing cycles, attribute conditions, transitive closures and recursive calls. Additionally, as these queries are executed rather frequently in interactive modeling applications, they have a significant impact on the runtime performance of the tool, and also on the end user experience.

In our framework to provide on-the-fly evaluation of constraints, we applied EMF-IncQuery, a state-of-the-art pattern matcher engine over EMF models based on incrementally maintained caches, resulting in (almost) instantaneous contract evaluation. More details are available in [28].

Without going into details the simplified example contract depicted in Figure 14 captures the condition that, “if there exist an *Application* with an *ApplicationInstance* (as the **pre**condition), then after the allocation step there cannot be *ApplicationInstances* that are not allocated to a

Partition (as the **post**condition)". For more details see [24].

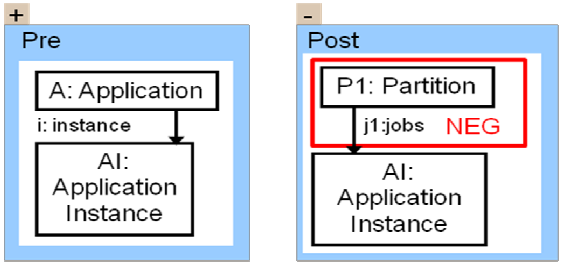

Figure 14: Contract for Application Allocation

Validation of end-to-end traceability

One key question in end-to-end traceability is to demonstrate that any *target element* can be traced back to its corresponding requirement. Showing this ability in a model driven development process can be problematic as separate part of models can be parts of the traceability (e.g., in our case the integration model is also part of the traceability) resulting in complex trace paths.

To solve this issue, our idea is to validate the existence of such trace paths through contracts and show that complete traceability is present in the whole mapping process from the PIADL down to the generated artifacts. The idea is based on the following assumptions: (i) defining the traceability relation, as a contract, between the input and output of a step is relatively simple, (ii) if a step is completed in the process and its contracts are validated then their postconditions can be treated as valid statements over the model, and finally, (iii) using the defined workflow of the development process all steps required for the generation of a configuration element can be followed back to its starting point (e.g., import step, creation step, etc.) allowing an induction based reasoning over the contracts of the traversed steps to show the existence of a valid traceability path.. However, it only proves the existence and does not provide the traceability matrix; future work is required to automatically generate it.

Related Work

There are numerous approaches in the literature introducing various model based techniques for the development of embedded system. Here we give a brief summary of some current EU research projects with significant relevance to design and verification of embedded systems involving model based techniques.

INDEXSY [15] aims to realize industrial implementations of cross-domain architectural concepts [17] developed in the *GENESYS* [29] project and give tool support based on MDD for its three target domains: *automotive*, *aerospace* and *railway.*

COCONUT [30] focuses on the definition of a formal framework [31] based on a tight integration of design and verification through refinement steps of an embedded platform design flow, from specifications to logic synthesis and software compilation.

TopCased [32] is an open source tool-kit (over the Eclipse platform) for the design, development and deployment of safety critical system using novel MDD techniques and support for languages like AADL and SysML.

CHESS [33] seeks to improve Model Driven Development practices and technologies to (i) better address safety, reliability and robustness functionalities as required by the aeronautical and railway industry and (ii) develop techniques to guarantee the correctness of assembled component embedded systems by reusing certification artifacts of the components used for the complete system.

FRESCOR [23] aimed to integrate advanced flexible scheduling techniques directly into an embedded systems design methodology, covering all the levels involved in the implementation, from the OS primitives, through the middleware, up to the application level using contracts to define the application requirements.

CHARTER [34] focuses on cost-reduction of certification of critical embedded systems by integrating real-time Java, model-driven development, rule-based compilation, and formal verification into a novel development process called *Quality-Embedded Development* (QED).

Conclusion and Future Work

Our approach demonstrated that Model-Driven Engineering can be effectively applied for the systematic development of ARINC 653 configuration tables. Additionally, we demonstrated the use of

model based validation techniques such as (i) complete end-to-end traceability from the high-level models down to the generated artifacts and (ii) model based on-the-fly validation of design contracts during the development process.

However, during the evaluation of the proposed technologies we have encountered gaps and shortcoming that point to future work and new research directions:

One key issue for the success of MDE in the safety-critical domain is the *certification of model transformation.* MT serves as the backbone of almost all model based technologies from code and model synthesis through model validation techniques to simulation. Up to date many work has been done regarding the V&V of transformations [35], however, certification issues were rarely covered in recent publications. Additionally, the complexity of tools may impose high qualification costs on tool vendors

As development of safety-critical system usually requires large number of developers the need for advanced collaborative support for the definition models like versioning, distributed development, access control etc. is becoming a key question [36][35].

Finally, MDE promises an easier way of integrating various tools based on a common integrated model (or model bus [32]) that allows their input and output models of the various tools to be treated in a common way. Additionally, it can give support for model based traceability a common requirement by various certification authorities.

References

[1] Airlines electronic engineering committee (AEEC), 2006, avionics application software standard interface - ARINC specification 653 - part 1 (supplement 2 - required services), ARINC Inc.

[2] The Object Management Group: System Modeling language, http://www.sysml.org/.

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

[4] Clarck, L., T. Ruthruff, B.. Hogan, , Development of Lockheed Martin's, F16 Modular Mission Computer Application Software using MDA http://www.omg.org/mda/mda_files/LockheedMartin.pdf.

[5] RTCA/EUROCAE, 1992, Software Considerations in Airborne Systems and Equipment Certification.

[6] Wind River, 2007, VxWorks 653 for Integrated Modular Avionics, Wind River White Paper, Alameda.

[7] GMV, 2009, SIMA Overview, GMV White Paper, Lisbon.

[8] The DIANA Project, Distributed, equipment Independent environment for Advanced avioNic Application, http://dianaproject.com.

[9] AUTOSAR Consortium: The AUTOSAR Standard, http://www.autosar.org/.

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

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

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

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

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

[15] The INDEXYS Project, INDustrial EXploitation of the genesYS cross-domain. architecture, http://www.indexys.eu/.

[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] Obermaisser, R., H. Kopetz (Eds.), 2009, GENESYS: A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems; Germany, SVH

[18] Schoofs, T:, 2010, The Use of SIMA in the DIANA Project. A Success Story, GMV White Paper, Lisbon.

[19] Parkinson, Paul, Larry Kinnan, 2006, Safety Critical Software Development for Integrated Modular Avionics. Wind River White Paper, Alameda.

[20] Felipe, Sérgio, 2007, ARINC 653 Validation Test-Suite Execution on VxWorks 653 2.1, Report Skysoft Portugal and Wind River, Lisbon.

[21] Schoofs, Tobias, 2009, ARINC 653 Validation Test-Suite Execution on VxWorks 653 2.2, Report GMV and Wind River, Lisbon.

[22] Wind River, 2007, Platform for Safety Critical ARINC 653 - Configuration Reference 2.2, Wind River Manual, Alameda.

[23] The FRESCOR project, Framework for Real-time Embedded Systems based on COntRacts, http://www.frescor.org/.

[24] Balogh, A., et al, 2010, Workflow-Driven Tool Integration using Model Transformation, Graph Transformations and Model-Driven Engineering, LNCS 5765, Springer.

[25] Ráth, I., A. Ökrös, D. Varró, 2010, Synchronization of abstract and concrete syntax in domain-specific modeling languages, Software and System Modeling, Spec. Issue on Traceability, Springer.

[26] Meyer, Bertrand, 1992, Applying ”design by contract”, Computer, IEEE, 25(10), pp. 40–51.

[27] Varró, D., A. Balogh: The model transformation language of the VIATRA2 framework, 2007, Science of Computer Programming 68(3), Elsevier, pp. 214–234.

[28] Bergmann, G., et al., 2010, ,Incremental Model Queries over EMF Models, In Proceeding of the 13th International Conference on Model Driven Engineering Languages and System, Oslo, Norway, Springer.

[29] The GENESYS project, GENeric Embedded SYStem Platform, http://www.genesys-platform.eu/.

[30] The COCONUT project, A COrrect-by-CONstrUcTion Workbench for Design and Verification of Embedded Systems, http://www.coconut-project.eu.

[31] Bloem R., et al., 2010, RATSY - A new Requirements Analysis Tool with Synthesis. Proc. of Computer Aided Verification (CAV), Edinburgh, Scotland, Springer, pp 425-429.

[32] TopCased, The Open Source Toolkit for Critical Systems, http://www.topcased.org/.

[33] The CHESS project, Composition with Guarantees for High-Integrity Embedded Software Components Assembly, http://chess-project.ning.com/.

[34] The CHARTER project, Critical and High Assurance Requirements Transformed through Engineering Rigour, http://charterproject.ning.com.

[35] Varró, Dániel, 2010, Towards Certifiable Model Transformations: A Survey, Budapest University of Technology and Economics, Department of Measurement and Information Systems, Technical report, Budapest.

[36] Bendix, L., Par E., 2009, Requirements for Practical Model Merge, In Proceeding of the 12th International Conference on Model Driven Engineering Languages and System, Denver, USA, Springer, pp. 167-180.

Acknowledgements

We would like to thank to Klaas Wiegmink, from NLR, for his help with air conditioning case study and Olivier Charrier, from WindRiver for his support on VxWorks.

This work was *mainly supported* by the EC FP6 DIANA (AERO1-030985) European project, however, the *validation of traceability by contracts research direction* was also partially supported by the Hungarian CERTIMOT (ERC_HU_09) project and the Janos Bolyai Scholarship.

Email Addresses

Ákos Horváth: ahorvath@mit.bme.hu

Dániel Varró: varro@mit.bme.hu

Tobias Schoofs: tobias.schoofs@gmv.com