An Industrial System Engineering Process Integrating Model Driven Architecture and Model Based Design

A. Sindico[1], M. Di Natale[2], and A. Sangiovanni-Vincentelli[3]

[1] Elettronica SpA, Roma, Italy, andrea.sindico@elt.it
[2] TECIP Institute, Scuola Superiore S. Anna, Pisa, Italy, marco.dinatale@sssup.it
[3] EECS Dept, University of California at Berkeley, alberto@eecs.berkeley.edu

Abstract. We present an industrial model-driven engineering process for the design and development of complex distributed embedded systems. We outline the main steps in the process and the evaluation of its use in the context of a radar application. We show the methods and tools that have been developed to allow interoperability among requirements management, SysML modeling and MBD simulation and code generation.

Keywords: System Engineering, Model-Driven Architecture, Model-Based Design, Platform-Based Design

1 Introduction

The complexity of modern cyber-physical systems is rapidly growing. Advanced system engineering methodologies are required to integrate all the competencies and specialty groups required for the realization of a system using a structured development process from concept to production to operation. The motivations and objectives of the industrial design process we present in this paper are:

- Improve the quality of the requirements moving towards their definition in a formal language and the need for tracking requirements into design artifacts and hardware or software implementations;
- Improve the quality of the functional solutions by early verification and validation on models using simulation, model checking or other forms of automated verification;
- Produce reusable and documented components at all levels in the design flow;
- Automatically derive implementations from models that are provably correct. Also, automatically generate documentations and possibly test cases.

Model-driven approaches such as domain-specific modeling languages, Model-Driven Architecture (MDA) and Model-Based Development (MBD) are possible choices to form the backbone of the design flow. Albeit MDA and MBD share the same principle (models as primary artifacts driving the design and development process), they differ substantially in their details. Each on his own is incapable of addressing all the challenges of modern system design. However, their strengths and weaknesses are complementary: MBD languages enable the realization of mathematical specifications that can be exploited for system simulation, testing and behavioral code or firmware generation, but they lack expressive power to represent complex system architectural aspects and execution platforms. Moreover, their extension mechanisms are quite limited in scope. On the other hand, MDA languages are very good at representing architectural aspects and are designed for being easily extended and provide mechanisms to transform models expressed in a language into another. However it is still hard to exploit these kind of models for model execution and simulation.

Starting from requirement capture, our approach follows the tenets of Platform-Based Design (PBD)[2], in which a functional model of the system is paired to a model of the execution platform. In particular, we present in this paper an integration of MBD and MBA to realize a comprehensive system engineering process based on the INCOSE framework [23]at Elettronica S.p.A (ELT)[1], one of the European leaders in the production of Electronic Defence equipment (EW). We describe tools and model integration techniques, automatic code generation using both MDA and MBD tools and the development of domain specific metamodels and profiles, extending the OMG MARTE standard.

2 The Design Process

2.1 Our Objectives and Motivations to Change

As any industry the main reasons why we have decided to evolve and update our design and development process are: to increase quality; to increase productivity; to reduce costs. In order to achieve these objectives we have identified some aspects of our design and development process that could be positively affected, for what concerns quality, productivity and costs, by a model driven approach. The first aspect is: documentation. As a company operating in the defense industry we have to provide a lot of documentation when designing and developing a product. According to the MIL-STD-498 standard [24], to which ELT is conform, more than 20 different documents have to be provided just for the design and development of the software architectgure of a system. For each document a review process has to be done in order to ensure its contents are correct, enough descriptive and coherent. This is a very expensive activity that has to be repeated every time we modify a project. Thanks to the model driven workflow here described we have managed to automatically obtain almost all the documentation from the models. The second aspect we wanted to optimize is the interaction among engineers with different specialties (i.e. SW, HW, FW) which work together but use different modeling and development tools. Often errors, due to misunderstanding, crept in and an extra effort was frequently needed in order to integrate their design activities. The process we present in this paper is instead a unified framework enabling SW,FW and HW engineers to share models and reduce integration effort. The third aspect is the reuse of components which is now optimized by the fact that the exploitation of modeling languages enable the definition of a functional architecture which is independent of a specific actual execution platform. Only when it has to be deployed onto a specific platform the functional model is related to a platform model. Such mapping eventually enables the automatic generation of the SW and FW artifacts which realize the functional model for that platform.

2.2 The Overall Structure

The model driven design and development process we have defined (shown in Figure 1) starts with the *System Requirements Analysis and Definition* stage that establishes functional and performance requirements. The output of this phase is a *System Subsystem Specification* (SSS) [24] document. Requirements are expressed, documented, managed and linked to test descriptions (for verification) using the **DOORS** tool by IBM. In order to provide the required formalization to requirements, DOORS textual descriptions are supplemented by **SysML** state and interaction diagrams, block diagrams, (interface) type definitions and OCL constraints. For the creation and management of the SysML models, the **Topcased** open source tool, based on the Eclipse Modeling Framework (**EMF**) is used. The need to preserve the consistency of the requirements and of the links tracking the requirements to architecture-level design decisions and (refined) subsystem requirements led to the development of automatic transformations between the DOORS and SysML tools and the information managed by them, implemented in a custom Eclipse plugin (shown as ① in the figure). The plugin leverages the support offered by EMF for the OMG languages for the development of metamodels (ECORE) to transform the DOORS requirements modules in an Ecore model, and model-to-model **QVT** transformations to keep the DOORS and SysML models synchronized.

The following stage of architecture-level *Solution Definition* translates these requirements into a system architecture design and the corresponding document called *System Subsystem Design Description* (SSDD) [24] that encompasses the functional and execution platform architecture, and addresses the functional requirements defined in the SSS. In the ELT process, DOORS is the master tool for storing the textual requirements associated with the SSS and SSDD. System models of the functional architecture and the execution platform are defined in Topcased, and the plugin connecting Topcased and Eclipse with DOORS synchronizes the architecture-level requirements in a similar way to what is done with the system-level specifications. In addition, the plugin detects refinement chains in the SysML model and automatically imports them as DOORS links (step ② in the figure).

System interfaces are described in additional documents: the *Interface Requirements Specification* (IRS)[24], and the *Interface Design Description* (IDD). The ELT process realizes a platform-based design approach [2] rendered here for the sake of simplicity as an early V&V model on top of a (conventional) V process.

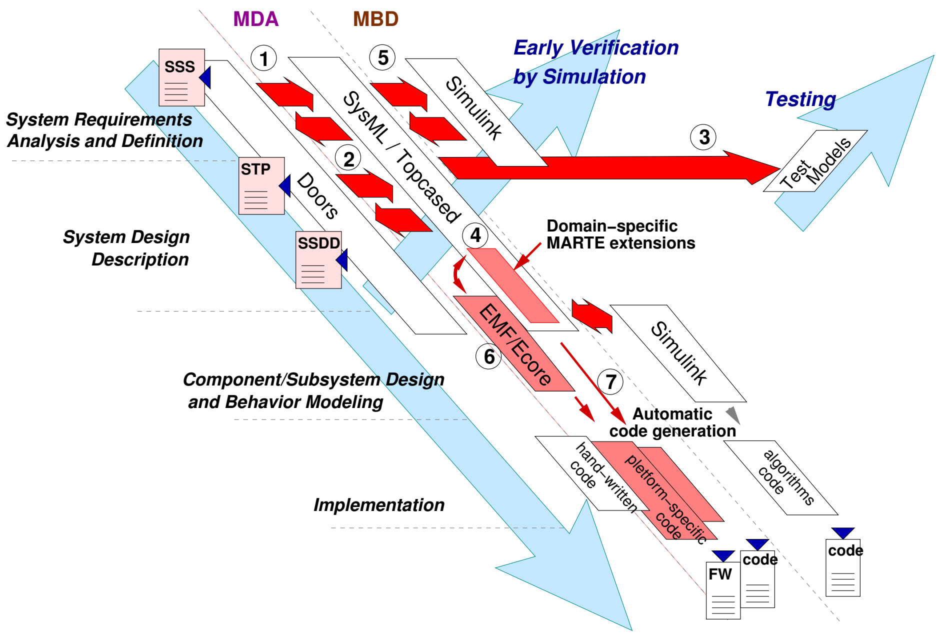
MDA
MBD
SSS
STP
SSDD
Doors
SysML / Topcased
Simulink
Early Verification by Simulation
Testing
Test Models
System Requirements Analysis and Definition
System Design Description
Domain−specific MARTE extensions
EMF/Ecore
Simulink
Component/Subsystem Design and Behavior Modeling
Automatic code generation
hand−written code
platform−specific code
algorithms code
Implementation
FW
code
code
1
2
3
4
5
6
7

Fig. 1. Early V&V, model transformations and automatic generation of implementations in the ELT process

Automation is provided for the system-level testing stage, where the SysML interaction diagrams defined for the system and subsystem specifications are automatically processed to generate the models of the system- and subsystem-level tests that verify them (③ in the figure). In architecture design, the SysML models of the system and the subsystems are defined according to the Platform-based Design paradigm, separating the functional model from the model of the execution platform, including the physical architecture. A third model represents the deployment of the functional subsystems onto the computation and communication infrastructure and the HW devices. To define the execution platform and the mapping relationships between the functions and the platform (which defines the model of the software tasks and the network messages, among others) domain-specific SysML extensions are required. We found that the standard MARTE profile [10] is not completely adequate for our needs. We therefore defined our own domain-specific stereotypes as extensions to MARTE (step ④ in the process). Synchronous reactive behavioral models of algorithmic components are developed in **MATLAB/Simulink**: a Mathworks toolset comprising a graphical modeling language (Simulink); a scripting language (MATLAB [18]) and a set of simulation, analysis and optimization and synthesis tools. These models provide an early validation of the system functionality by simulation and are used to automatically derive a software or firmware implementation. The Simulink models can refine functional subsystems of the SysML architecture (in a top-down development) or provide building blocks for the definition of the system (in a bottom-up flow). To support both paths, we developed scripts and code generation templates that can be used to transform a Simulink subsystem (hierarchy) into a (set of) SysML block(s) and viceversa, preserving the flow specifications at the interface ports (step ⑤). The process uses code generation techniques or automatic generation of firmware implementations starting from the Simulink models for the behavioral part, and from the SysML architecture description for the implementation of the communication and synchronization functions and for the code framework of the software tasks ⑦. The synthesis of the communication functions over shared memory and serial links uses a definition of the message model based on an Ecore metamodel that has been defined *ad hoc* ⑥.

2.3 Structure of a Project

Figure 2 depicts the reference SysML project structure used to organize and relate the model elements used in the system design process. The project consists of six packages:

- a *SystemRequirements* package containing a SysML model of the system requirements imported from a DOORS SSS module by means of a RIF export;
- an IntefaceDataTypes package containing a SysML model defining the Data Types and Interfaces provided and required by the system and its parts;
- a *SystemFunctionalArchitecture* package containing a SysML model describing the functional architecture of the system, as a network of subsystems exchanging data signals;
- an *ExecutionPlatform* package containing a SysML model describing the execution platform in terms of the HW and basic software components, including boards, memories, processing units (cores), network connections, but also device drivers, operating system(s) and middleware. For this purpose, we extended the standard MARTE profile [10] for real-time and embedded systems, providing baseline concepts for representing HW/SW systems;
- a *Mapping* package containing a SysML model using an extension of the MARTE Mapping profile to specify how functional components and behaviors are mapped onto an execution platform, generating the software architecture of tasks and messages. To guarantee independence and reusability as well as visibility of the design entities involved in the mapping, the mapping model imports both the functional and platform models;
- a *Test* package containing a SysML model defining all the tests by means of which the system requirements shall be verified.

Separated from the project models we maintain a domain model which is shared among the different projects and contains domain specific meta-entities describing the information managed by the system. This organization enables the reuse of Interfaces and Data Types and according to the PBD paradigm, allows deploying (by mapping) the same functional model into different platforms of execution.

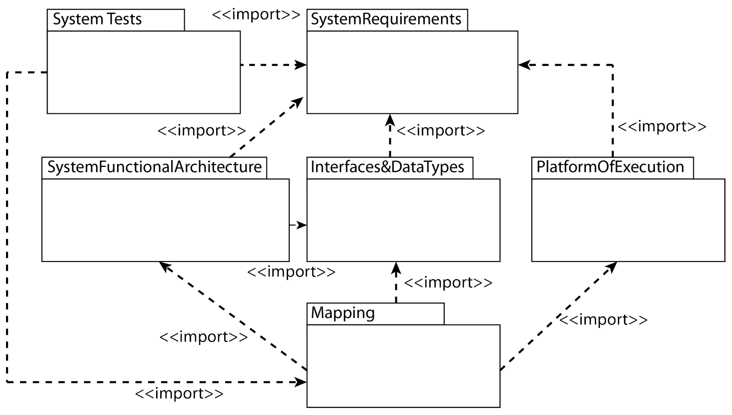

Fig. 2. The structure of SysML projects in ELT

2.4 System Requirement Modeling and Management

Our process starts with the definition of the system requirements in DOORS. Requirements are expressed in natural language, which can be easily understood by customers and other stake-holders, but is subject to inconsistencies, omissions and duplication of information. To partially obviate to these problems, the SSS requirements are paired with a SysML (semi)formal description. Next, when the architecture models are defined, each model artefact must be traced to the requirements that originated it and also to the (subsystem-level) requirements that it defines.

To this end, we realized an Eclipse plugin for automating the exchange of information and the synchronization of requirements models and diagrams between DOORS and Topcased. The plugin exploits a standard XML format for requirements interchange called RIF (*Requirement Interchange Format* [25]). A metamodel for the RIF format is available in Ecore and used in our approach to automatically generate an Ecore model by importing the RIF XML exported from DOORS. As with any other Ecore model, the Eclipse modeling framework automatically generates an editor that can be used to modify the imported data.

In addition, we built a synchronization engine, based on correspondence rules between the DOORS RIF objects and their attributes (in a given module) and corresponding SysML elements and attributes in Topcased. The synchronization module is based on rules written as QVT Model to Model transformations [17] to synchronize the content of corresponding elements or automatically generate them when requested. On top of the synchronization component, a wizard allows the user to create correspondences between sets of requirements and sets of SysML elements. In this way, parts of a source DOORS module (i.e. paragraphs, interface requirements, functional requirements, parametric requirements) can be imported, or updated, into a new or already existing target SysML model. Among the predefined rulesets provided by the wizard, the user can take a DOORS module containing SSS requirements and automatically import it into a (newly generated) SysML project as a set of SysML requirements. These SysML requirements are used in the subsequent phase to define *Satisfy* relationships with the SysML model entities of the architecture design (SSDD).

2.5 Computational Independent Models

A *Computation Independent model* or CIM (also called domain model) is a conceptual model of the domain of interest (or problem domain) which describes the various entities, their attributes, roles and relationships, plus the constraints that govern the integrity of the model. Using Ecore, ELT defined an electronic warfare meta-model (details can be found in [20]) as a formal and structured representation of the electronic warfare concepts, ranging from the concept of *Platform* (i.e. aircraft, ships, tanks, etc.), to *Sensors* (i.e. Radars, ESMs, etc.), electromagnetic *Waveforms* (i.e. Radiofrequency, Pulse Repetition Interval, etc.) and *Countermeasures* (i.e. Jammers, Chaff, Flare, etc.). This Ecore model represents our Computational Independent Model and we want to keep it unique for all the system we design. Our first aim in designing a new system is thus verifying whether our domain model is expressive enough to cope with the system requirements. If the domain model does not contain concepts or entities' characteristics referenced in the system's requirements we extend it or adapt the requirements (in collaboration with the customer).

2.6 Defining the (Platform Independent) Functional Architecture

Once the system requirements are defined and the ontology of the domain entities is available in the CIM, we start the actual system design by defining the functional architecture of the system. To this end, we use SysML with the MARTE (Modeling and Analysis of Real Time Embedded Systems) profile [10]. The SysML functional model of the system is a network of subsystems. Each SysML *Block* represents a (sub)system functionality defined independently of the eventual implementation technology (i.e. Software, Firmware, etc.) or the HW upon which it will be executed (i.e. CPU, GPU, FPGA, etc.) according to the PBD paradigm. The *System Functional Architecture* package contains the SysML design of the architecture, consisting of Blocks, representing functional subsystems, ports (interaction points) and connectors among ports. As in any SysML model, standard (synchronous) invocation of services is modeled through UML *StandardPorts* each typed with a UML *Interface* providing *Operations*, each of which represents a behavior of the related component invoked synchronously (in a blocking fashion) with respect to the caller. Non-blocking communication may occur according to a discrete-event model (through signal events) or according to a stream of data values produced periodically accoding to a discrete-time base. In the case of (asynchronous) signal events, we use ports stereotyped in MARTE as *ClientServerPort* that allow transmission or reception of UML *Signals*. The corresponding port interfaces are stereotyped as *ClientServerSpecification*. A UML *Reception*, stereotyped as *ClientServerFeature* may be associated to each signal received by the port, specifying the behavior

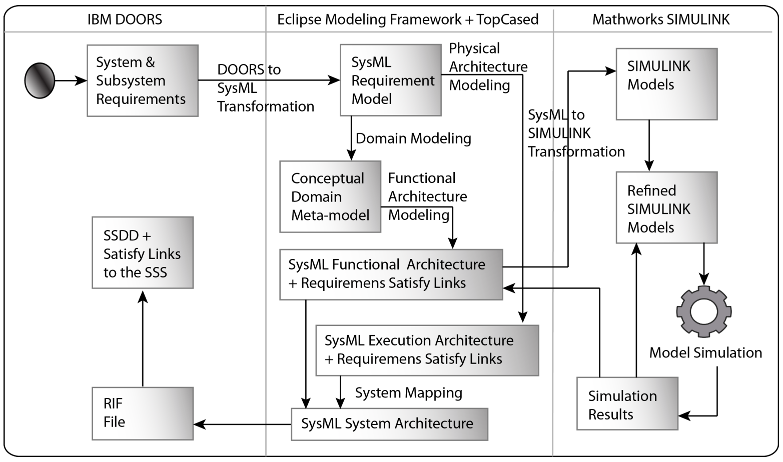
IBM DOORS
Eclipse Modeling Framework + TopCased
Mathworks SIMULINK
System & Subsystem Requirements
DOORS to SysML Transformation
SysML Requirement Model
Physical Architecture Modeling
SIMULINK Models
SysML to SIMULINK Transformation
Domain Modeling
Conceptual Domain Meta-model
Functional Architecture Modeling
Refined SIMULINK Models
SSDD + Satisfy Links to the SSS
SysML Functional Architecture + Requiremens Satisfy Links
SysML Execution Architecture + Requiremens Satisfy Links
Model Simulation
System Mapping
Simulation Results
RIF File
SysML System Architecture

Fig. 3. The system-level design flow

response to it. The case of communication through periodic data streams is of special interest, because it mirrors the communication semantics that is used in the Synchronous Reactive (SR) models produced by the Simulink tool. In this case, the behavior of a SR functional subsystem needs to be further restricted. Communication ports will be SysML flow ports stereotyped as *SRFlowPort*. Also, an SR subsystem is stereotyped as *SRSubsystem* and characterized by the realization of a standard runtime interface consisting of a single *Step* method. The *SRSubsystem* stereotype is associated with an execution period attribute. Interfaces and client server specifications are contained in the *Interface and Data Types* package so that they can used multiple times in different contexts. Data types are generated from the previously described domain model. This guarantees that the system relies on well structured and homogeneous data descriptions, each constrained within the value range prescribed by the SSS. The functional model of the system is part of the SSDD description and must be linked to the SSS requirements from which it originated. After being imported from DOORS, the SSS requirements are mirrored in the SysML tool as a set of SysML requirements. Each component, port, interface and signal of the SysML functional model is connected to the requirement it satisfies by means of the SysML *Satisfy* relationship. OCL scripts verify that each functional component satisfies at least one requirement. Each subsystem will define a set of derived requirements on its structure and behavior. Those requirements are linked with a *Trace* relatonship to the functional subsystem (or element, like a port or even a connection) that originates them. By following the chain SSS Requirement $\rightarrow$ *Satisfy* $\rightarrow$ SysML element $\rightarrow$ *Trace* $\rightarrow$ SSDD Requirement, the tool is capable of inferring a *Derive* Relationship between the SSS and the SSDD requirements. The synchronization plugin with DOORS allows to define other associations (QVT transformations) for the automatic generation of the objects in a DOORS SSDD module starting from a set of SysML architecture design descriptions and requirements. Further, the tool generates a DOORS refinement link for each *Derive* relationships between SSS and SSDD requirements in Topcased. Figure 4 depicts a portion of a real functional architecture model extracted from an actual ELT system. The figure illustrates the kind of complexity in terms of interactions and interfaces that is typical of ELT models.

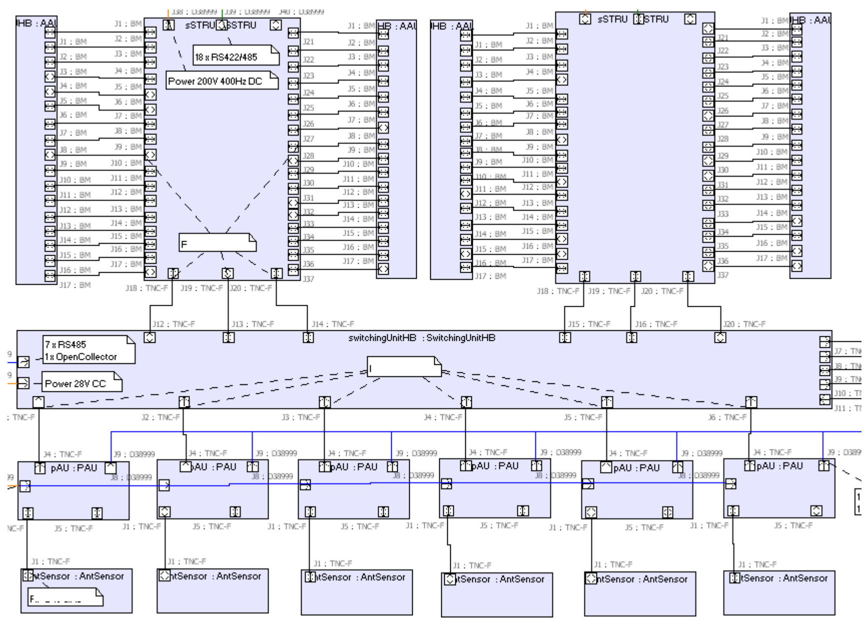

Fig. 4. A portion of the Internal Block Diagram of the EW Integrated System

Once the functional architecture is defined, a state diagram must be associated to every subsystem in the functional architecture model. For each state transition we define both triggers (e.g., the reception of a signal on a port) and guard conditions by means of OCL constraints. After, the behaviors associated with each of the subsystem states are defined.

The UML Superstructure [7] defines two kind of behaviors: *executing* and *emergent*. Interaction diagrams (i.e. sequence diagrams) are used to model emergent behaviors. Communication among active objects occurs through UML *Signals*. Active objects hide their *Operations*, which are invoked only upon the reception of a signal. This approach improves the separation of concerns. While modeling emergent behaviors we add time constraints for each request/response exchange or operation call by means of UML *DurationConstraints*.

Some executing behaviors are defined by activity diagrams and then implemented manually in C++. Other actions define behaviors that are imported from a Simulink model or for which an MBD development flow is going to be used. These are stereotyped as *Analytical* and must refer to a Step function of an *SRSubsystem* block. In case of a top-down process, the development flow makes use of transformations from the interface view of the SysML *SRSubsystem* block into the specification of a Simulink Subsystem, complete with its ports and datatype specification as Bus Objects. The Simulink subsystem is then further developed in the Mathworks environment. The transformation currently in use is an Acceleo script (described in [21]) that transforms the SysML block into a set of MATLAB scripts. More often, however, the functionality to be developed has already been prototyped in Simulink and a reverse transformation generates a SysML block. In this case, a MATLAB script generates an XML file compliant with an Ecore metamodel developed *ad hoc* for the representation of Simulink subsystems in EMF. A QVT model-to-model transformation then generates the SysML block from the Ecore model. Later, at code generation time, special care must be taken when generating the data implementations of the port interfaces and the calls to the Step operations of the *SRSubsystem* blocks.

2.7 Execution Platform Modeling and Mapping

The execution platform and the mapping models define the structure of the HW and SW architecture that supports the execution of the functional model. For both models we leveraged the standard definitions of the MARTE profile. However, it was apparent that MARTE is extensive and general and yet still lacks several features of interest.

The execution platform is defined in a package called *PlatformModels* with the same principles of hierarchical decomposition used in the functional model. Here, blocks represent hardware components at different levels of granularity, but also classes of basic software, including device drivers, middleware classes and operating system modules. The MARTE profile provides several concepts for the basic software classes, but is unfortunately not adequate for the definition of hardware components. For example, not a single stereotype is dedicated to the representation of physical network links (of any type), connectors or cables. Also, concepts like message frames (for Ethernet, Controller Area Network or other standards) and the placement of data signals onto frames are missing. For this reason, we had to define our own taxonomy of stereotyped definitions for most hardware components. Most of them were quite straightforward, others revealed minor complexities or subtleties for usability, such as for the definition of broadcast buses, which are derived by extension from the connector and block metaclasses to ensure the possibility of representing one-to-many connections (impossible with SysML connectors), but at the same time, allowing a more natural modeling of physical links with connectors whenever possible. The definition of stereotypes for other physical elements proved to be much more difficult, as is the case of connectors with multiple pins, a subset of which realizes a communication bus (such as, for example, an Ethernet link). In this case, the connector stereotype cannot be simply obtained by extending the port concept, because ports cannot contain other ports representing pins. Figure 5 shows an example of UML profile we have developed in order to model serial(e.g., RS232, RS422, ARINC-429, and USB) and parallel (e.g., PCI, and VME) busses and cables. The Mapping definitions are contained in the *MappingModels* package, in which

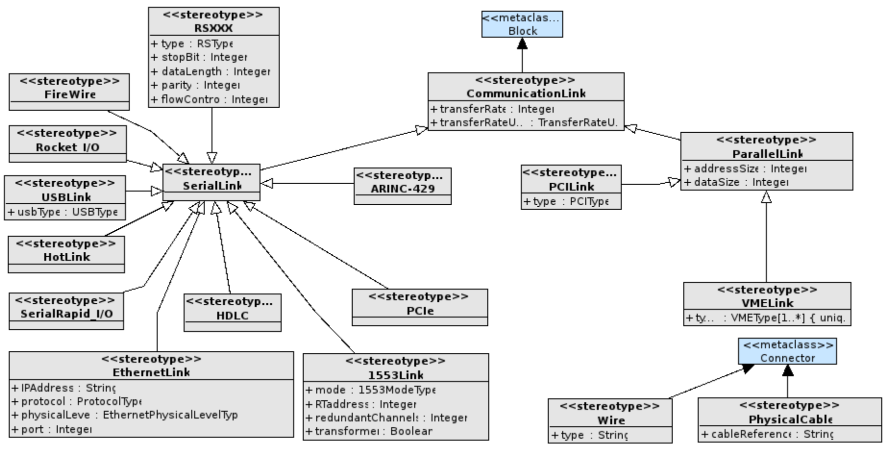

Fig. 5. A UML profile involved in physical communications

the mapping of the functional subsystems onto the execution platform generates the SW architecture of tasks, messages and logical resources. For the definition of concurrent software we use the *SWConcurrency* package which defines the stereotype *SwConcurrentResource* to represent entities competing for computing resources and executing sequential instruction segments. The elements of this package provide an execution context (e.g., stack, interrupts enable/disable and registers) for an execution flow (sequence of actions).

2.8 Automatic generation of documentation, system and unit tests and software/firmware implementations

A significant improvement in the productivity and quality of the process is obtained using the tools for the automatic generation of models, documents and implementations (code and firmware). ***Documentation*** Once the elements of the Functional Architecture and the Execution Platform models have been put in a mapping relationship, we can apply a transformation workflow aimed at automatically generating an SSDD document describing the system architecture. Most of the SSDD, SRS and SDD [24] contents are, in fact, already present in the defined functional and platform models and can be imagined as different views of the same models. The QVT transformations of the SysML-to-DOORS plugin generate a RIF file that is imported, generating DOORS modules for the SSDD. IRS and IDD specifications and the corresponding documents, each linked to the related SSS requirements, according to what specified in the SysML model with the *Derive* links.

Test cases

QVT transformations have been defined to process the interaction diagrams describing the system-level behaviors and generate the sequence diagrams of the test suites that verify out of range and in range invocations for each constrained operation call. Additional sequence diagrams are generated for all the duration constraints aimed at testing out of time conditions. The models representing system and unit tests are generated from the constraints on the entities of the functional architecture and can be an input to further transformations aimed at the automatic generation of the related *System Test Plan* and *System Test Description* documents.

Implementation generation

The automatic generation of the implementation of functionality is performed using Mathworks tools (Simulink Coder) to derive the FPGA (if firmware) or C (C++) code implementation (if software) of the behavior of some functional subsystems. The generated FPGA implementation communicates with the other subsystems using a set of registers and shared memory locations. The C or C++ generated code follows the conventions of the code generator (two functions for the subsystem initialization and termination and a function with a conventional name for the runtime evaluation of the block outputs given the inputs and the state). The Simulink Coder conventions also define the names of the variables implementing the interface ports. Other functional subsystems are developed manually by hand-written code or purposely designed HW or firmware.

For the infrastructure that provides communication and synchronization among blocks, we exploit the possibility of generating code using Acceleo transformation rules (this work has not been completed as yet). For the communications among subsystems implemented by hand-written C++ code, there are two options. In the case of local communication (detected from the mapping information of the SysML model), we defined a set of transformations generating boost [27] active objects, signals and state machines according to what specified in the platform independent architecture models. In this case, data structures are also automatically generated and OCL constraints turned into run-time checks on the specified value boundaries. When communication is remote (that is, when the mapping model places the two communicating subsystems on different nodes) and, in case the connecting network is of Ethernet type, we make use of a purposely developed Ecore model that defines mapping details, such as on what (TCP/UDP/IP) message the data signal is transmitted and with which offset and encoding, to generate automatically the network interface part of the communication.

For the communication between hand-written C++ code implementing functional subsystems and subsystems generated in SW from Simulink model, Acceleo scripts automatically generate the wrappers that provide the marshalling of parameters to the variables implementing the input ports and retrieving the data from the output port variables. The Step function implementing the subsystem runtime behavior is invoked in the context of a software thread executing at the appropriate rate.

Finally, for the case of communication between automatically generated firmware implementations and software subsystems, the read and write interface operating on the shared registers and memory locations is automatically generated using Acceleo based on the information provided in an additional Ecore mapping model, defining the position of data signals in memory and/or HW registers.

2.9 Dealing with Legacy Systems and Sub-Systems

The described process is applicable to the design and development of any new system though particular attention has to be provided when dealing with already existing legacy components (either HW, SW or FW) for which no models are available. In this case our approach is to manage the legacy component as a black block and define modeling elements wrapping it. To this end we apply the *Facade* design pattern [47] each time the component does not provide a cohesive enough well structured architecture. Such solution can be permanent, for those component which are very stable and are not supposed to be extended/improved in the future, or temporary, for those components we think will need to be extended or improved in the future. In this second case we may in fact decide to start a reverse engineering process aimed at obtaining a set of SysML models describing the component itself. This decision is any time taken after an evaluation of the required extra cost compared to the return of investment that could derive.

3 A case study on an electronic warfare system

As an example of what can be achieved with our process we present results related to a project developed at ELT from 2010 to the end of 2011 called Electronic Warfare Manager (EWM [29]). The EWM realizes a Mission Computer for an electronic defence suite capable of gathering information from the available sensors (i.e. Radar Warning Receivers, Laser Warning Systems and Missile Warning Systems [28]), providing an integrated situation assessment that decides what of the available electronic countermeasures (i.e. Chaff, Jammers, etc.) to apply and managing their execution. From an industrial point of view, a fundamental requirement for the system is that it must ensure connectivity with different physical communication links on different platforms, according to the customer. Retargetability of communication interfaces was obtained by the automatic generation of IRS, IDD, and related C++ implementation from an abstract SysML model of the interfaces. By modeling each communication protocol as described in Section 2.4, more than 15K lines of code (LOC) and almost 200 pages of documentation (IRS and IDD) have been automatically generated. The code implementing these transformations is about 2,4KLOC implying that the effort has been significantly reduced. Also the remaining part of the system architecture has been modeled, as described in section 2.4, by means of SysML, MATLAB and Simulink with the automatic generation of the corresponding implementations for additional 25KLOC. Although it is in principle possible to generate almost all the required code, as of now we still have to add some glue code allowing interactions among generated components. For the EWM, this code was about 5KLOC so that the whole system is about 45KLOC, 90 percent of which has been automatically generated from models. In order to have an evaluation of the saved effort we used the Constructive Cost Model II (COCOMO II) [30] [32]. A web application provided by the Center of Systems and Software Engineering (CSSE) [31] estimates for a SW project of 40KLOC (the amount of code we automatically generated), leaving all the other COCOMO parameters as nominal, a development effort of 169.9 Person-months for a cost of 1868638$. This does not mean that 90% of this amount was actually saved, because the Inception and Elaboration phases, corresponding to the System Requirement Analysis and System Design Phases, are still (mostly) manually performed. However, the design step that is most affected by automatic code generation is the Construction phase, which is also the most expensive (estimated by the CSSE to be $1,420,166 for a duration of 12.5 months and 10.3 people involved). Accuracy of the CSSE-COCOMO estimates was confirmed from the fact that the Inception and Elaboration phase estimates turned out to be quite close to the actual effort and cost experienced at ELT.

4 Related work

The amount of work related to our project is simply staggering. We provide some references with respect to and technologies used in the stages of the process, but there are surely many more that are omitted. The match of a functional and execution architecture is advocated by many in the academic community (examples are the Y-cycle [33] and the Platform-Based Design PBD [2]) and in the industrial domain (the AUTOSAR automotive standard [34] is probably the most relevant recent example) as a way of obtaining modularity and separation of concerns between functional specifications and their implementation on a target platform. The OMG and the MDE similarly propose a staged development in which a PIM is transformed into a Platform Specific Model (PSM) by means of a Platform Definition Model (PDM) [35]. The development of a platform model for (possibly large and distributed) embedded systems and the modeling of concurrent systems with resource managers (schedulers) requires domain-specific concepts. The OMG MARTE [10] standard is very general, rooted on UML/SyML and supported by several tools. MARTE has been applied to several use cases, most recently on automotive projects [37]. However, becasue of the complexity and the variety of modeling concepts it has to support, MARTE can still be considered an ongoing work, being constantly evaluated [36] and subject to future extensions. Several other domain-specific languages and architecture description languages of course exist, such as, for example EAST-AADL and the DoD Architectural Framework. Several other authors [38], [39] acknowledge that future trends in model engineering will encompass the definition of integrated design flows exploiting complementarities between UML or SysML and Matlab/Simulink, although the combination of the two models is affected by the fact that Simulink lacks a publicly accessible meta-model [38]. Work on the integration of UML and synchronous reactive languages [40] has been performed in the context of the Esterel language (supported by the commercial SCADE tool), for which transformation rules and specialized profiles have been proposed to ease integration with UML models [41]. With respect to the general subject of model-to-model transformations and heterogenous models integration, several approaches, methods, tools and case studies have been proposed. Some proposed methods, such as the GME framework [42] and Metropolis [43]) consist of the use of a general meta-model as an intermediate target for the model integration.Other groups and projects [44] have developed the concept of studying the conditions for the interface combatibility between etherogeneous models. Examples of formalisms developed to study the formal conditions for compatibility between different Models of Computation are the Interface Automata [45] and the Tagged Signal Language [46]. In this context our contribution is to provide an example of an actual industrial framework in which different tools and languages (i.e. DOORS, UML, SysML, MARTE, DSLs, SIMULINK, M2M and M2T Transformations,etc.) are integrated together into a single design and development workflow. Our contribution aims to show that there is not a "*ready to use*" model driven process suitable to any industry. It is instead necessary to design the process itself and properly tailor it around specific needs that could vary from a company to another. In our experience this process engineering activity can only be performed by a team with a very deep and wide knowledge of the existing technologies and methodologies together with a strong understanding of the company domain and needs.

5 Conclusions

In this paper, we presented an industrial flow and related tools featuring the integration of Model Driven Architecture and Model Based Design methodologies using a Platform-Based Design paradigm for the realization of military real-time embedded systems conforming to the MIL-STD-498 standard. The process is characterized by integration of heterogeneous languages, methods and tools, from requirements to implementation generation, in a flow in which the backbone is provided by the open source Eclipse Modeling Framework (EMF) and its metamodeling, model-to-model and model-to-code transformation capabilities. We provide system traceability from DOORS requirements to SysML design elements and system-level tests, and viceversa. In our process, the functional architecture is developed separately from the execution platform and later merged with it, according to the PBD paradigm. The development of the models for the execution platform revealed inadequacies and limitations of the standard MARTE profile, which was suitably extended and for which a new release that addresses the concerns regarding the communication modeling is strongly advocated. In our approach, almost all MIL-STD-498 compliant documentation is automatically generated from system models. Similarly, on selected case studies, about 90 percent of the target code and firmware implementations are generated from models with substantial savings. Behavioral code is generated from Simulink models, while infrastructure, communication code and tasking code is developed from SysML models and Ecore models, processed by Acceleo scripts.

References

1. Elettronica S.p.A.: http://www.elt-roma.com
2. Sangiovanni-Vincentelli, A., ”Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design”, Proceedings of the IEEE, Vol. 95, No. 3, pp. 467-506, Mar. 2007.
3. The Object Management Group: http://www.omg.org
4. Mukerji,J.,Miller,J., ”Overview and Guide to OMG’s Architecture”, http://www.omg.org/cgi-bin/doc?omg/03-06-01
5. Paterno,F. ”Model-Based Design and Evaluation of Interactive Applications, Springer-Verlag”, London, 1999
6. The Meta Object Facility (MOF): http://www.omg.org/spec/MOF/2.4.1
7. The UML Superstructure: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/
8. The UML Infrastructure: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/
9. The System Modeling Language: http://www.sysml.org/docs/specs/OMGSysML-v1.1-08-11-01.pdf
10. Modeling Analysis of Real Time Embedded Systems (MARTE) profile: http://www.omg.org/spec/MARTE/1.0/PDF/
11. The Eclipse Modeling Framework: http://www.eclipse.org/modeling/emf/
12. P. Popp, M. Di Natale, P. Giusto, S. Kanajan, C. Pinello, Towards a Methodology for the Quantitative Evaluation of Automotive Architectures, Proceedings of the Design Automation and Test in Europe Conference, Nice, April 15-18 2007
13. Q. Zhu, Y. Yang, M. Di Natale, E. Scholte and A. Sangiovanni-Vincentelli, ”Optimizing the Software Architecture for Extensibility in Hard Real-time Distributed Systems”, IEEE Transactions on Industrial Informatics, 2010, vol 6 issue 3
14. TopCased: http://www.topcased.org
15. Acceleo: http://www.acceleo.org/pages/home/en
16. MOF Models to Text Transformation Language: http://www.omg.org/spec/MOFM2T/1.0/
17. Query View Transformation Language: http://www.omg.org/spec/QVT/1.0/
18. MATLAB: http://www.mathworks.it/products/matlab/
19. SIMULINK: http://www.mathworks.it/products/simulink/
20. Sindico, A., Tortora, S. Chiarini Petrelli, A., Fasano, M.V., ”An Electronic Warfare Meta-Model for Network Centric Systems,” Cognitive Information Processing (CIP), 2010
21. Sindico, A., Di Natale, M., Panci, G., ”Integrating SysML With SIMULINK Using Open Source Model Transformations”, SIMULTECH 2011: 45-56
22. IBM DOORS: http://www-01.ibm.com/software/awdtools/doors/
23. System Engineering Handbook: http://www.incose.org/ProductsPubs/products/sehandbook.aspx
24. The MIL-STD-498 Standard: http://www.letu.edu/people/jaytevis/Software-Engineering/MIL-STD-498/498-STD.pdf
25. The Requirement Interchange Format: http://www.omg.org/spec/ReqIF/1.0.1/
26. The Object Constraint Language: http://www.omg.org/spec/OCL/2.0/
27. Boost: http://www.boost.org/
28. S. A. Vakin, L. N. Shustov, R. H. Dunwell, Fundamentals of Electronic Warfare, Artech House Radar Library, 2001.

29. Tortora, S., Sindico, A., Severino, A., "A Data Fusion Architecture for an Electronic Warfare Multi-Sensor Suite", Cognitive Information Processing (CIP) , 2010
30. Bohem, B., Clark, B., Horowitx, E., Westland C.,Madachy, R., Selby, R., "Cost models for future software life cycle processes: COCOMO 2.0", Annals of Software Engineering, Vol. 1, Num. 1, pp. 57-94.
31. The Center of Systems and Software Engineering:
32. Boehm,B.,"Software Engineering Economics", Englewood Cliffs, NJ:Prentice-Hall, 1981. ISBN 0-13-822122-7
33. B. Kienhuis, E. F. Deprettere, P. v. d. Wolf, and K. A. Vissers, "A methodology to design programmable embedded systems - the y-chart approach," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*. London, UK: Springer-Verlag, 2002, pp. 18–37.
34. "Autosar, specifications 4.0," http://www.autosar.org/, 2010.
35. Stephen J. Mellor, Scott Kendall, Axel Uhl, Dirk Weise, "MDA Distilled" Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA 2004
36. Ali Koudri, Arnaud Cuccuru, Sebastien Gerard, Franois Terrier "Designing Heterogeneous Component Based Systems: Evaluation of MARTE Standard and Enhancement Proposal". Proceedings of the MODELS Conference 2011, pages 243-257
37. Ernest Wozniak, Chokri Mraidha, Sebastien Gerard, Franois Terrier: "A Guidance Framework for the Generation of Implementation Models in the Automotive Domain". EUROMICRO-SEAA 2011: 468-476
38. Y. Vanderperren and W. Dehaene, "From uml/sysml to matlab/simulink: current state and future perspectives," in *Proceedings of the conference on Design, automation and test in Europe*, DATE '06, Leuven, Belgium, pp. 93–93.
39. D. B. F.I.T.T., "Eda survey results," 2005.
40. A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, Jan. 2003.
41. G. Berry and G. Gonthier. "The synchronous programming language ESTEREL: Design, semantics, implementation". Science of Computer Programming, 19(2), 1992.
42. G. Karsai, M. Maroti, A. Ledeczi, J. Gray, and J. Sztipanovits, "Composition and cloning in modeling and meta-modeling," *IEEE Transactions on Control System Technology (special issue on Computer Automated Multi-Paradigm Modeling*, vol. 12, pp. 263–278, 2004.
43. F. Balarin, L. Lavagno, C. Passerone, and Y. Watanabe, "Processes, interfaces and platforms. embedded software modeling in metropolis," in *Proceedings of the Second International Conference on Embedded Software*, ser. EMSOFT '02. London, UK: Springer-Verlag, 2002, pp. 407–416.
44. J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, Y. Xiong, "Taming Heterogeneity—the Ptolemy Approach," Proceedings of the IEEE, v.91, No. 2, January 2003.
45. L. de Alfaro and T. Henzinger, "Interface automata", Proceedings of the 8th European Software Engineering Conference, Vienna, Austria, 2001
46. A. Benveniste, B. Caillaud, L.P. Carloni, and A. Sangiovanni-Vincentelli, "Tag Machines", Proceedings of the ACM International Conference on Embedded Software (EMSOFT'05), Jersey City, NJ, USA, Sep. 2005.
47. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, Mass., 1995