2

Experiencing Model-Driven Engineering for Railway Interlocking Systems

Fabio Scippacercola[1,2], András Zentai[3] and Stefano Russo[1,2]

[1]DIETI, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy
[2]CINI-Consorzio Interuniversitario Nazionale per l'Informatica, Italy
[3]Prolan Process Control Co., Szentendrei út 1-3, H-2011 Budakalász, Hungary

2.1 Introduction

For a company to be competitive in the market, following technologies and being updated with new trends and practices, is essential. In safety-critical domains, the introduction of new practices and methodologies is slower than in other engineering fields, since safety standards and long established practices tend to defer the adoption of new emerging technologies, until assessments and time reveal them mature and safe. Slow introduction of new methods is especially characterizing the railway domain, where the lifespan of products could easily reach decades or even a century. Now it is long time that Model-Driven Engineering (MDE) techniques and tools have been proposed, but their maturity – especially for safety-critical systems – is still debated.

Some recent surveys investigated the adoption of MDE methodologies and technologies in practice [1, 2]. They revealed the increasing adoption of MDE in industry. The technology is attractive for the development of critical systems, since it can speed up the activities of Verification and Validation (V&V), and it enables the early verification of systems, through techniques such as model reviews, guideline checkers, Rapid Control Prototyping (RCP) and Model- and Software-in-the-Loop Tests. These techniques shift the cost

of development from the phases of V&V to the ones of requirement analysis and design, thus leading to benefits in terms of residual errors. Companies not performing model-in-the-loop testing find almost 30% more errors during module test [3].

Prolan Co. is a Hungarian company, which develops certified products for safety critical process control and rail signaling systems. Prolan joined the European project "CErtification of CRItical Systems" (CECRIS [4]); in its framework, Prolan started an industrial-academic partnership for the transfer of knowledge of MDE techniques from the academy to the company, with the goal of assessing their level of maturity for industrial adoption.

During this activity, it emerged the lack of well-defined processes for the development of a CENELEC SIL-4 safety critical signaling system that was suited for the real industrial needs.

2.2 Background: MDE

As for most engineering branches, advances in software engineering have always resulted from increases in the level of abstraction. Let us consider, for instance, one of the most peculiar activities of this discipline, namely computer programming: the first abstraction, i.e., the second generation programming languages – or assembly languages – were born soon after programmers had struggled with binary machine code; then came the third generation programming languages (procedural and object-oriented), that freed the programmers from low-level details of the machine, and then fourth generation languages, which added more facilities and masked recurrent problems, such as the representation of data and the interworking between heterogeneous systems. The same holds in other areas, such as operating systems, middleware technologies, and network protocols. In this perspective, MDE aims at raising the level of abstraction in software design and verification [5], and promises to change the traditional methodologies of software development.

Model-driven approaches focus on a model, i.e., on a set of specifications or representations of a system that neglect aspects that are not of interest at the current stage in a software process; the process advances transforming the model in documents, intermediate artifacts, or in the final product. The result is that MDE shifts the traditional development paradigm, based on different kinds of artifacts composed by domain experts in multiple formats, to a common formalism – the model – by which the artifacts are obtained through computer-assisted transformations. This model-centric paradigm provides

several benefits, leading to increased productivity and quality of artifacts, shorter development time, and enhanced automation, which includes automatic code generation and automatic support to the software engineering activities.

Since models have always been applied at different extents in engineering problems and activities, there are many acronyms with fuzzy borders in the universe of software engineering. We refer to the terminology of Brambilla et al. [6].

When processes exploit models as support for their goals they are part of Model-Based Engineering (MBE), and we call the activities document-centric, since models are only a means to achieve the targets, but there is no particular emphasis on them. Therefore, MBE is the broadest term, encompassing all the methodologies and activities that employ models.

Model-Driven Engineering focuses on the processes where models are key artifacts of the activities (model-centric). When we restrict to considering MDE for supporting the development of systems, we can use the more specific term of Model-Driven Development (MDD). One approach of MDD is the Model-Driven Architecture (MDA), proposed by the Object Management Group (OMG) [7]. The Model-Driven Testing (MDT) is a theory of software testing that introduces concepts enabling to transform models in test-cases in order to support V&V activities. Even though MDT is not an OMG standard, it uses an OMG's standard profile, the UML-Testing Profile (UTP) [8, 9].

Model-Driven Engineering is founded on concepts of *models* and *transformations*: instead of producing (textual) documents as artifacts – requirements, design, code, test artifacts – engineers focus on models as primary artifacts.

Models are defined in (semi-)formal languages, which are typically machine-understandable and drawn with the support of tools. Other artifacts are derived through defined transformations, be they: Model-to-Model transformations (M2M) or Model-to-Text transformations (M2T) from models to textual documents, source code or testing artifacts (such as test cases and test scripts).

As argued by Kent [10], MDE can identify different levels of decomposition and can employ ad hoc or domain-specific languages for models and transformations, whereas MDA is bound to OMG's standards.

The OMG is an international open membership not-for-profit consortium grouping many IT companies and organizations around the globe. OMG first conceived MDA as a technology to overcome the interoperability problems of applications partially addressed by the CORBA standard [11]. Indeed, even if CORBA provided a good solution for the interoperability of applications,

it became soon clear how it is difficult for large enterprises to standardize on different middleware platforms: enterprises have applications on different middleware, that have to be integrated even though this process turned out to be expensive and time-consuming. Furthermore, middleware systems continue to evolve and even CORBA could not be a guarantee for next decades. Therefore, MDA was proposed as a better way to reach portability, interoperability and reusability through architectural separation of concerns in the OMG vision that postulates how the myth of a standalone application or standard for developing software as well as for data interchange died.

The recent version 2.0 of the Guide to the standard [7] defines MDA as an approach for deriving value from models and architecture in support of the full life cycle of physical, organizational and IT systems. MDA became an approach to deal with complexity and interdependences in large systems, namely to derive value from modeling by defining the structure, semantics, and notations of models using industry standards.

In order to enable (automatic) transformations of models, mechanisms were introduced to reason on the models themselves: this has been done through the concept of meta-modeling, namely introducing models for modeling languages. These concepts are commons to MDE, but MDA standardized the formalisms to use, so as to have four layers of abstractions:

- M0 is the user data layer, it is the layer at lowest abstraction and the elements are concrete objects of the problem domain.
- M1 is the layer of modeling concepts. Here are the UML models of entities that abstract the user data layer, like UML classes or association. At this level are models defined by software engineers to define the requirements or architecture of the system.
- M2 is UML Metamodel, i.e., M2 defines, through UML, the syntax of UML models in M1, as well as their semantic. For instance, M2 will constraint you to do not use UML links for connecting classes but UML objects. M1 models can be seen as instances of concepts of M2 layer and, by M2, you can check consistence of your UML models.
- M3 is most abstract layer defined by OMG. At this level is Meta-Object Facility (MOF) language. By MOF OMG can define syntax and semantic for meta-languages. In the MDA, MOF enables to define transformation rules among different models (of M1 layers) that are compliant to different meta-models (of M2 layers).

Using its modeling infrastructure, it is possible to define rules to transform models into other models (M2M) or model into text (M2T). With M2T

transformation, MDE refers specifically to that kind of transformation that produces source code (or other textual documents) from models.

2.2.1 MDA Viewpoints and Views

Model-Driven Architecture starts with the well-known and long-established idea of separating the specification of the operation of a system from the details on how that system uses the capabilities of its platform. MDA enables to specify a system independently from the platform that supports it, and to transform the system specification into one for a particular platform.

A *viewpoint* specifies a reusable set of criteria for the construction, selection, and presentation of a portion of the information about a system, addressing stakeholder concerns [7]; in other words, a viewpoint defines the abstractions to adopt to focus on particular concerns within the system. A *view* is a representation of a system that conforms to a viewpoint [7].

In MDA terms, abstraction eliminates certain elements from the defined scope and may result in introducing a higher-level viewpoint at the expense of removing detail. A more abstract model encompasses a broader set of systems, whereas a less abstract model is more specific to a single system or restricted set of systems. One important capability of MDA is the automation that provides for the transformation between levels of abstraction by the use of patterns.

Model-Driven Architecture specifies three viewpoints, which offer levels of separation of concerns to realize a system. The three viewpoints are:

Computation Independent Viewpoint (CIV). The computation independent viewpoint focuses on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined;

Platform Independent Viewpoint (PIV). The platform independent viewpoint focuses on the operation of a system while hiding the details necessary for a particular platform;

Platform Specific Viewpoint (PSV). The platform specific viewpoint combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.

The recent version of MDA standard [7] reduces the emphasis on the CIV, and defines a platform as a set of resources on which a system is realized. This set of resources is used to implement or support the system. For instance, a platform can be the organizational structure or a set of buildings and machines

(in case of business or domain platform types); or operating systems, programming libraries, and CPUs (when considering computer hardware and software platform types).

A platform model also specifies requirements on the connection and use of parts of the platform, and the connections of an application to the platform. Example: OMG has specified a model of a portion of the CORBA platform in the UML profile for CORBA. This profile provides a language to use when specifying CORBA systems. The stereotypes of the profile can function as a set of markings. A generic platform model can amount to a specification of a particular architectural style.

Considering the previous views, MDA defines the Computation independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM). MDA refines CIM in PIM and in PSM using model transformations during development process.

2.3 The Maturity of MDE

Several surveys analyzed the diffusion and the benefits of Model-Based and Model-Driven techniques and technologies into industrial practices, after 30 years from the introduction of the first MD tools on the market. However, these analyses are still not enough to get a complete picture about the state of the MD practices. Indeed, an aspect that is often neglected by these surveys is that the utilization of MD techniques is tightly dependent on the domain, which influences the demands of the users as well as the stability of the environment and the availability and maturity of the supporting tools. The domain of embedded systems, for instance, has seen the diffusion of sophisticated MD tools such as Matlab Simulink or SCADE, that keep evolving in the offered functionalities since they were introduced on the market, many years ago. However, if we consider all other domains, we can see that the adoption of MDE in software companies differs from that in the domain of embedded systems: although MDE is always perceived beneficial, the benefits are not as evident as in the embedded system industry.

In general MDE seems not completely mature yet, and the feasibility of its adoption partially debated, with tools not enough stable integrated, and much of the MDE potential yet to be demonstrated. Summarizing the various observations in the surveys of past years in industry, we may conclude that:

- *MDE is spreading in industry, but it is still far to be pervasive*. It followed the concurrent evolution of modeling languages (such as UML)

and of related tools: in 2005 practitioners were using MBE for conceptual modeling [12], in 2008 model-centric approaches were perceived better than code-centric ones in most of tasks [13], in 2010 and 2011 MDE has been observed in a wide range of application domains [14–21], despite there are many problems and no general and common consensus on these approaches;

- *Models are mainly used for design and documentation*, while the benefits of advanced techniques (such as code generation, test case generation, or model animation) are lowly exploited: models are introduced mostly as an enabling technology inside the process, to enable business that otherwise would not be possible [14–16];
- *UML is gaining popularity, but support tools are not enough mature yet to build toolchains meeting the specific needs of companies*: they are considered one of the biggest problem by the industry, that is worried about ease of their usage, the vendor lock-in problem, and the interoperability among different tools [18–21];

Model-Driven Engineering depends on the business domain and on organizational factors, and its adoption requires changes in the personnel skills, the software processes, and the company practices. MDE demands for special skills and for changes in the roles of developers and software engineers: retraining programmers to think at a higher level of abstraction can reveal a difficult task. These aspects have not been well addressed so far, and the current approaches do not adequate to the people, but the people have to adapt to them.

A partially different scenario is observed in the domain of embedded systems, where we can draw the following picture:

- Model-based techniques are widely adopted (almost pervasive in automotive domain), and models are used not only for informative and documentation purposes but they were the key artifacts of the development processes [1, 2].
- The needs for introducing models was mainly for shorter development time, and to improve reusability and quality, whereas less than half had the need to introduce models to exploit formal methods, or because they were required by the standards [1, 2].
- The activities of V&V had a huge impact by their adoption in the automotive domain [3]: the automotive industry was used to exploit model-driven approaches for the early verification of the systems, by techniques such as model reviews, guideline checkers, RCP and

Model- and Software-in-the-Loop Tests, that lead to better quality, reduced development time, due to the shifting of the costs to the phases of requirement analysis and design;

- According to [22], UML is not used widely due to short lead-time for the software development, or lack of understanding or knowledge of UML models; however this survey, limited to MDE/MDA in the Brazilian industry, does not agree with [1, 2] targeting the European industries of embedded systems. These authors found that the majority of survey participants were using Matlab/Simulink/Stateflow, followed by Eclipse-based tools. The most used modeling languages were the OMGs ones (UML and SysML);
- As for generic software companies, in the top shortcomings identified there are the scarce interoperability and usability of tools, and the high (initial) effort to train developers [1, 2].

Why was the diffusion of MB and MD techniques different in the embedded systems domain with respect to other application areas? We claim that this is due to:

i. The different weight of the activities in the development process (more emphasis on design and implementation for generic software systems; more emphasis on analysis and V&V for embedded systems);
ii. The parallel evolution of the code-centric technologies that are available for the development, which raised even more the level of abstraction during the design, and simplified the way the systems are implemented. The hypothesis partially reflects the different focus on the adoption of models in the two domains, since there is more emphasis on design and documentation in the general market, and on the V&V techniques for the embedded systems.

The cited surveys identify the current state of the adoption of MD techniques in industry by collecting the opinions of the practitioners on the benefits and drawbacks of model-based and model-driven techniques. However, besides these quantitative data, there is the need of empirical studies that analyze qualitatively and critically the merits and faults of model-driven approaches. Indeed, the success or failing factors of MDE are still unclear, and more research is needed [23].

A systematic review of empirical studies on MDE from 2000 up to June 2007 was performed by Mohagheghi and Dehlen [24]. They show that MDE can effectively reduce the cost and development time, however this depends on the grade of adoption in the development process: a success story is the

one of Motorola [25, 26], that used MDE for more than 15 years in a wide spectrum of activities, ranging from protocol implementations up to handheld devices or network controllers; they experienced an increase in quality and productivity (ranging from 1.2× to 8×) and an approximately 33% reduction in the effort required to develop test cases.

Motorola could achieve these results within a mature process that was supported by own-made translators and tools for the model exploitation. Indeed, one common issue of MDE is the absence of well-defined processes [24, 27, 28], as the application of MDE requires changes in the activities, corporate culture and skills of the employees: many software engineering methods are not fitted to use models as main artifacts, and the environments seems not mature enough. Some previous studies attempted to apply pre-existing processes to MDE, or to create own ones, but MDE shifts the importance of many activities to (automatic) transformation rules, and change consolidated development process is not a naive task. The study [29] reports a successful introduction of a MBE process after 4 years and three projects had been defined and consolidated: there is the need to look beyond the technical benefits of a particular approach to MDE and instead concentrate on social and organizational issues [16].

Moreover, the process becomes a more difficult problem in safety-critical domain, where compliance with certification standards poses additional requirements on the methodologies for product life cycle. For these kind of systems, the major part of costs are for the activities of V&V, so rigorous and well-assessed techniques have to be integrated within the development process for the early detection of faults and to guarantee the quality of the product. In addition, non-functional requirements, such as safety, reliability and timing requirements, are a primary concern that have to be taken into account by these processes: current MDE methodologies do not cope with stringent functional requirements and qualities in current systems, i.e., the ability of these approaches to adapt to rapidly changing hardware and implementation platforms that are highly complex [23].

Parallel to the challenge of the product life cycle, there is the open problem of the supporting tools: they are not mature yet, and influence most of the adoption of MDE. Moreover, the vendor lock-in problem is also perceived as a problem, and the companies prefer to adopt open source solutions or to develop their own tools. Indeed, the tools are not well usable, do not interoperate between themselves, do not keep in synchronization the models at different level of abstractions, are not flexible to collaborative working, and are not suited with the adoption of different models and modeling

notations [23]. Thus, model-driven processes have to carefully consider the problem of defining the toolchain for supporting the activities.

2.4 A Model-Driven Methodology for Prolan

In the period when CECRIS started, Prolan was developing the next generation of railway interlocking systems, and in particular the first product of this generation, the *Prolan Block* (PB), a safety-critical system for railway interlocking that must be CENELEC EN 50126, EN 50128 and EN 50129 SIL-4 certified.

The system is deployed alongside railway segments, which are named *blocks*. Each block is equipped with a PB, with sensors for detecting incoming and outgoing trains (these sensors are the axle counters), and with semaphores that are part of the signaling system. The PB manages the block (Figure 2.1), receiving data from sensors, and properly setting the semaphores according to its internal state.

The interlocking is realized by the overall distributed system that consists of interacting PBs, which must ensure that no collision will happen on the railway, directing the train movements by proper sequences of signals. For instance, according to the specific regulations, the yellow lamps can indicate that the next block's semaphore is red because there is an obstacle (e.g., a train) in the block after the next (e.g., there is a train two semaphores ahead).

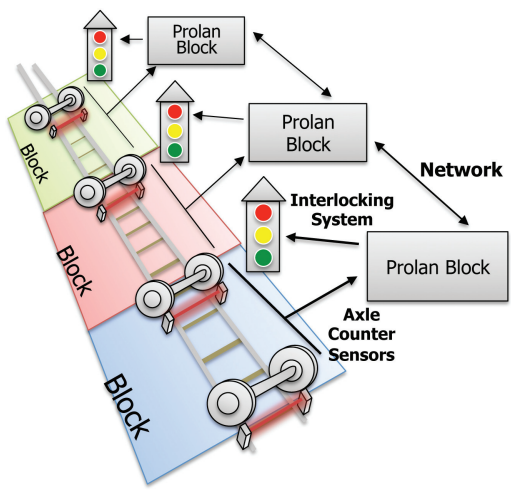

Figure 2.1 A representation of the Prolan Block and its operating environment.

Prolan was interested in understanding the potentialities that model-driven technologies could give for the development of the PB and for the other products of the same generation. Indeed, small and medium size enterprises like Prolan are interested in model-driven technologies but there are barriers to their introduction, for the deep changes that these require into the organization and in the current industrial practices. Indeed, for the adoption of MDE Prolan needs to carefully rethink and redesign its current product development life cycle, that currently complies with the railway standards CENELEC EN 50126, EN 50128 and EN 50129, as well as the skills of the employees, even if no proven-in-use model-driven lifecycle for this domain is available and supported by long-term evidence.

The traditional Prolan development life cycle follows the V-Model and is compliant with the European railway standard EN 50128. The activities of the CENELEC V-Model process can be grouped in those concerning development, that are on the left side of the 'V', and those focusing on V&V, that are on the opposite side as it can be seen in Figure 2.2. The activities of V&V require planning stages that are performed before their actual execution: these planning stages are carried out during design.

Besides the activities in the V-Model, CENELEC EN 50128 also prescribes requirements on the documents produced at each stage, as well as on the project organization. For instance, if we consider the highest integrity level (SIL-4), distinct people have to test, verify and validate the product, in order to cross-check their work. The phases adjacent to the 'V', the Software Planning and Software Assessment, aim at tuning and assessing the activities of the life cycle, defining the tasks to be performed during the process and checking that the product and all artifacts satisfy the requirements and comply with the standard.

To gain experience on model-driven technologies, Prolan started a collaboration with CINI in the framework of the CECRIS Project to develop a development process enhanced with model-driven approaches.

Since Prolan wanted a concrete and feasible option for replacing its current methodology, the researcher proposed a development process backed to Prolan's traditional process. This solution minimizes the impact of the change on the organization, and is also compatible with the safety standards pursued by the company. The adaptation of the development processes of Prolan to MDE focused on core phases of the CENELEC V-Model, starting from the System Development Phase up to the Software Validation Phase, i.e., on the Software Development Life Cycle (SDLC).

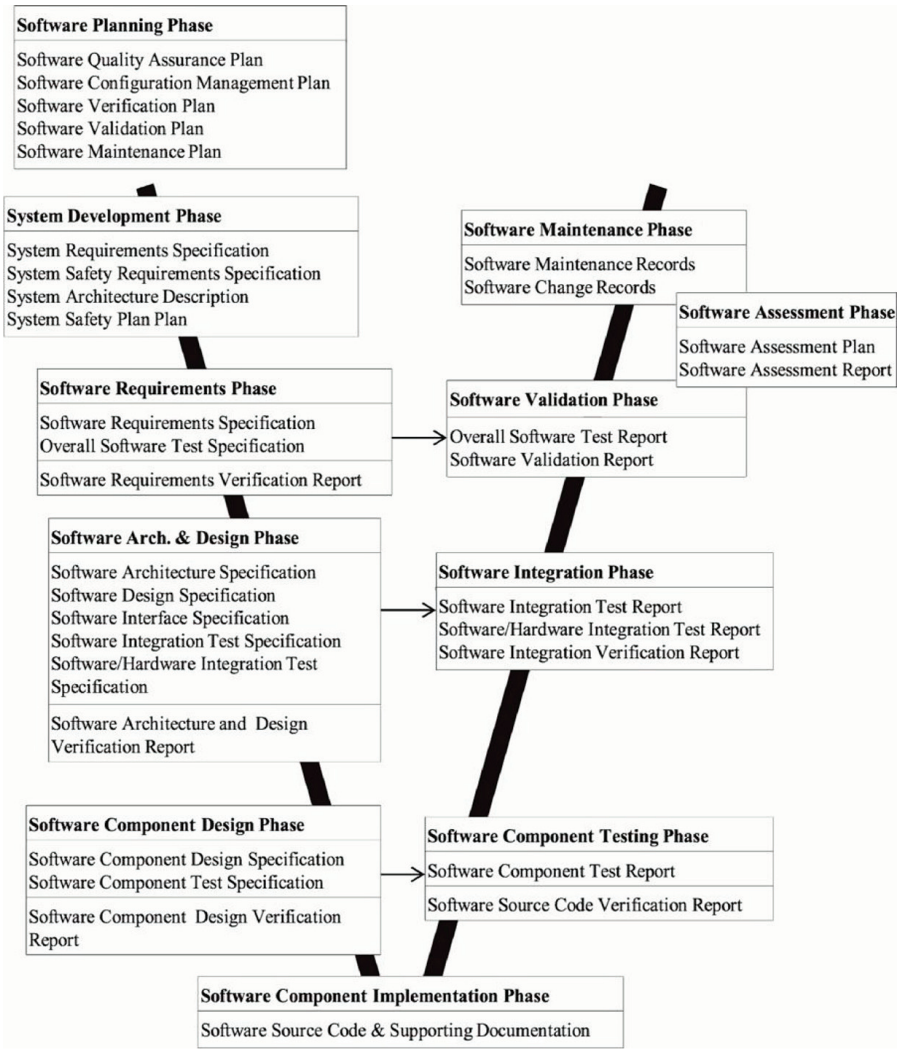

Figure 2.2 Software Development Life Cycle according to EN 50128.

The proposed model-driven V-Model lifecycle is shown in Figure 2.3: it is composed of a left, center and a right part; the title lines of the boxes refer to the SDLC activities performed by Prolan, according to CENELEC EN 50128 standard. For each activity, the boxes contain the models produced, and the formalisms used. Arrows represent dependency between the artifacts. The Component Design also depends on the Component Verification Design if it exploits the test model to early detect faults.

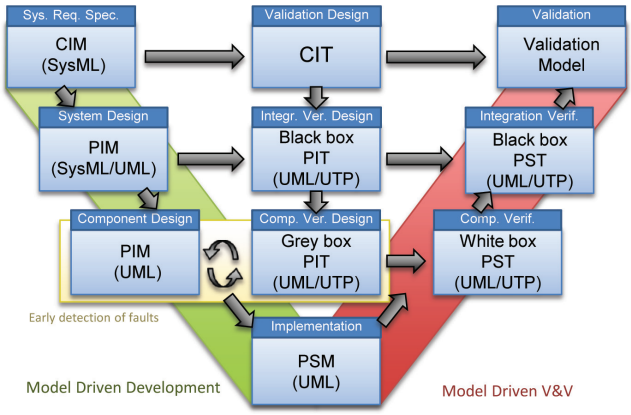

Figure 2.3 The adapted model-driven V-Model life cycle for Prolan [31].

On the left there are forward engineering activities (system analysis, design and implementation); the phases in the center are for V&V planning, while on the right side there are the activities of V&V execution. The CENELEC V-Model life cycle adopts implicitly different viewpoints on the system for each level of the 'V': the top level focuses on the system as a whole, the level below uses a viewpoint on the system architecture, then it considers the components and their internal design; finally, the lowest level of the 'V' sees source code details. These abstractions are used on both sides of the V-Model, for development and V&V.

The activities are assigned to a number of roles that comply with the CENELEC EN 50128 standard. We consider the following roles and responsibilities:

- The **Requirements Manager** is responsible for specifying the software requirements. (S)he shall be competent in requirements engineering and be experienced in application's domain (as well as in safety attributes);
- The **Designer** transforms software requirements into a solution, defining the system architecture and developing component specifications. (S)he has be competent in the application area, and in safety design principles;
- The **Implementer** transforms design solutions into data, source code or other representations to create the product software artifacts. (S)he has to be competent in engineering of the application area and implementation languages and supporting tools;
- The **Tester** develops the test specifications, and performs the test execution. (S)he has to be competent in the domain where testing is carried out;

- The **Integrator** manages the integration process using the software baselines, developing the integration test specification. (S)he has to be competent in the domain where component integration is carried out.

All these roles require advanced modeling skills, and experience with MDE, as well as with the adopted formalisms and tools.

The process starts with System Requirements Specification, by defining the system environment and software requirements. Then, System Design and Component Design are carried out. The former defines a high-level system architecture, identifying the hardware-software interface, and the components interfaces. Requirements are then allocated to components, and the Designer specifies their responsibilities and expected interactions. Finally, in Component Design the Designer completes the components with the internal design, and the Implementation concludes the development.

For enabling forward engineering to model-driven technologies, we define in three stages a CIM, a PIM, and PSM, following the MDA principles.

The V&V planning activities (Validation Design, Integration Verification Design, and Component Verification Design) have been isolated at the center of the V-Model. They are followed by the ones of V&V execution that are performed on the right side of the 'V', i.e., Validation, Integration Verification and Component Verification. For instance, Validation Design produces the Overall Software Test Specification after the System Requirement Specification. Then, the actual validation is performed in the Validation activity, at the end of the 'V', after Integration Verification, to assess the product conformance to requirements.

For the phases of V&V, we propose a model-driven methodology based on the MDA abstractions: the planning phases use Platform Independent Test Models, whereas the execution phases build Platform-Specific Test Models. In fact, the V&V execution phases on the right side of the 'V' benefit from the availability of the implementation, which constrains the technological platform.

Using this methodology, Prolan aims at improving the reuse of artifacts of design and V&V, supporting most of activities of the life cycle with model-driven approaches. Prolan wanted to evaluate the adoption of OMG standards, i.e., SysML [30] and UML, to be open to multiple tools and promote the interoperability of the models. It is worth to note that custom profiles can be introduced in the process to potentiate the automatic generation of artifacts throughout the whole SDLC, thus reducing the manual efforts.

We remark that since Prolan's products must undergo safety certification, one of the main requirements of the methodology is to exploit model-driven

technologies for supporting multiple activities of V&V. Indeed, the proposed process is open to multiple forms of V&V, and includes techniques of early system validation, through the definition of the Computation Independent Test (CIT) model.

2.4.1 Experimentation within A Pilot Project

Prolan started a pilot project on a subset of requirements for the Prolan Block, in order to assess the benefits and drawbacks of the model-driven technologies.

2.4.2 System Requirements Specification

At this phase, the Requirements Manager defines the system and the specification of software requirements. We defined a CIM starting from the high-level system specification.

The CIM models requirements, and the relations between them, in SysML, because the language turns out particularly suited in this phase due to the Requirement diagram, the Use Case Diagram, and the Block Definition Diagram. In particular, SysML Requirement Diagram is useful to display textual requirements, and their relationships, and to trace them with other modeling elements.

Prolan built in the pilot project a CIM using MagicDraw [32], a modeling tool created by No Magic. Functional and non-functional requirements of the system were described using requirement diagrams meanwhile the system context was described with block definition diagrams. Modeling using the MagicDraw tool was introduced in an earlier phase of the CECRIS project in the framework of the knowledge transfer with Budapest University of Technology and Economics. Using models to capture requirements increased requirement quality significantly because the graphical representation made it possible to overview complex systems as well as the constraints of the modeling environment forced the engineers to create consistent requirements. We found it convenient to separate functional and non-functional requirements in different groups as well as to mark derived requirements using the refinement relationship.

An example of functional and non-functional requirements modeling can be seen in Figures 2.4 and 2.5. As it can be seen from Figure 2.6, the PB system is connected to a Radio Block Center (RBC) to a PB Human Machine Interface (HMI) to a Station Interlocking System and to track occupancy

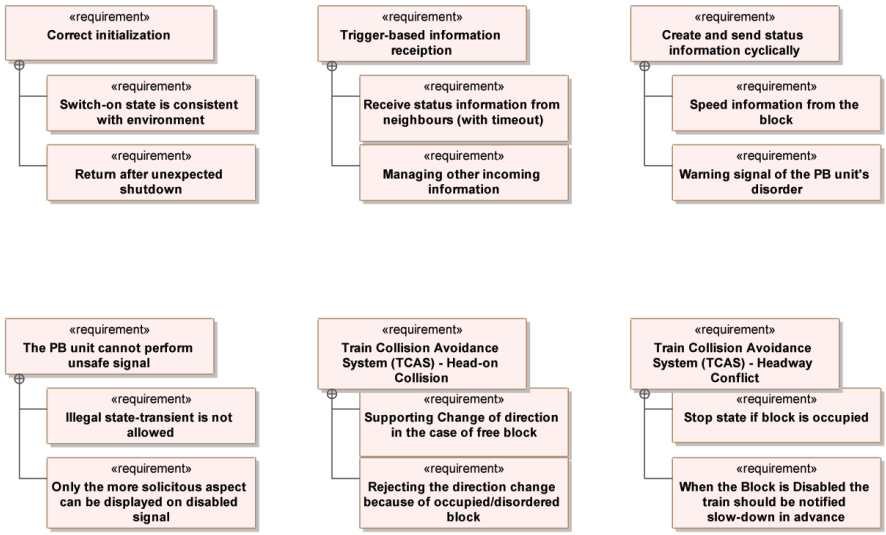

Figure 2.4 *Prolan Block* (PB) functional requirements.

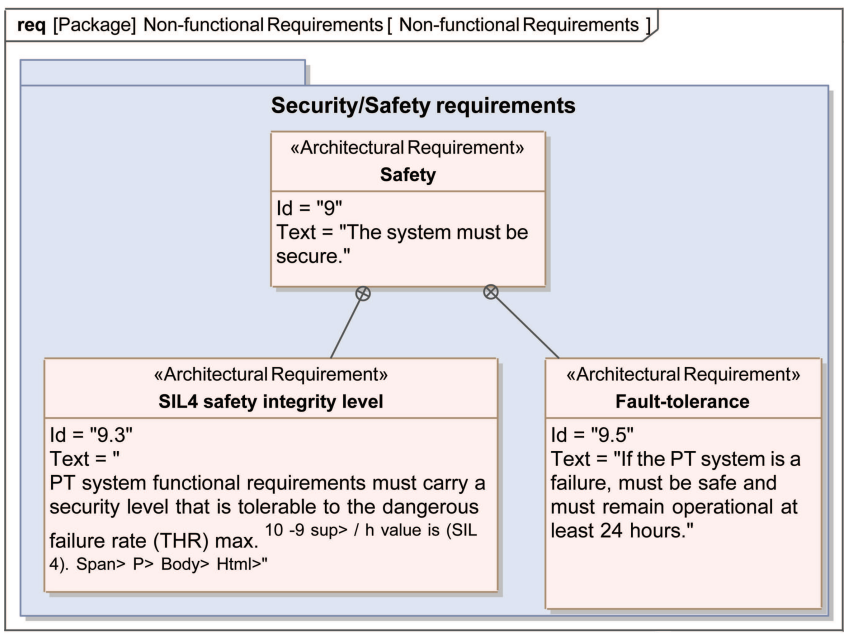

Figure 2.5 PB non-functional requirements.

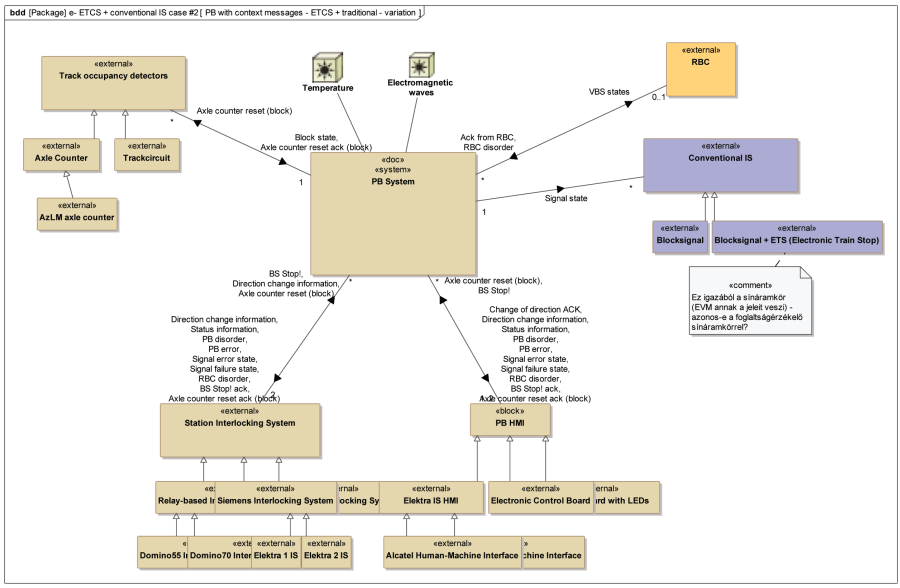

Figure 2.6 BDD diagram showing the environment of the PB.

detectors. In the BDD diagram not only the related components but also the multiplicity as well as the exchanged information and signals could be visualized.

Not only other actors, but also their relations and compositions were modeled with the BDD. Use case diagrams were created to describe in which functionalities the actors are involved (Figure 2.7).

One use case of the PB HMI is to receive the status of the PB and display it. Another use case is to reset the track occupancy detectors in case the operator activates the axle counter reset.

High-level functionalities of the system defined by functional requirements and use cases are further detailed by behavioral diagrams: state machine diagrams, activity diagrams and sequence diagrams. Requirements coming from the railway domain like the description of the semaphore's behavior (in Figure 2.8) are primarily introduced into the CIM.

Active support from the CINI side was necessary in modeling the CIM, because the technology was relatively new to the requirement engineers of the company. During this phase 6 SysML Requirement diagrams; 12 SysML Block Definition diagrams; 41 Use Cases diagrams; 6 State Machines diagrams; 29 Activity diagrams; and 33 Sequence diagrams were created.

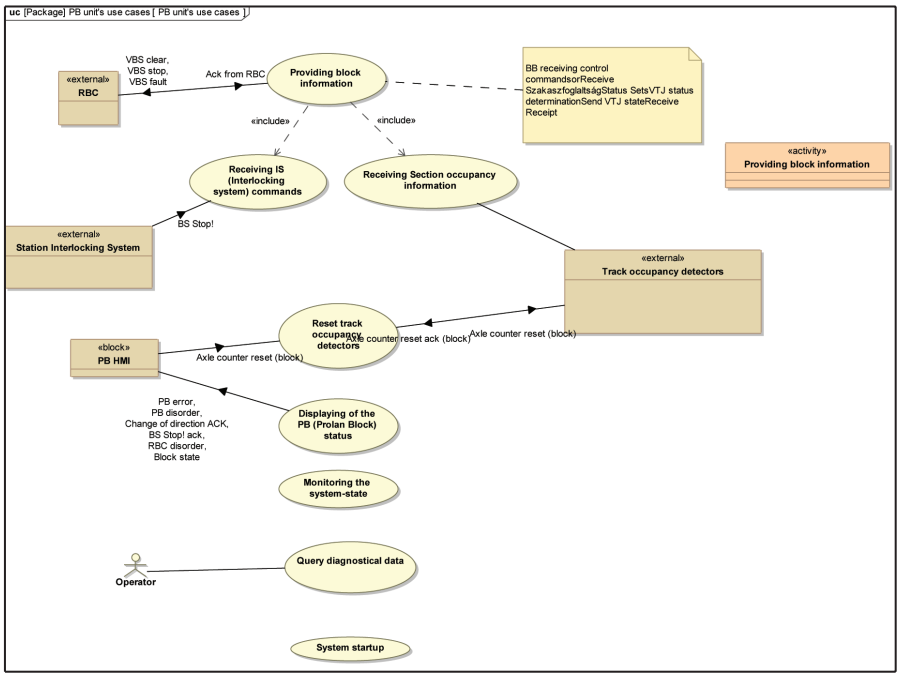

Figure 2.7 Computation Independent Model (CIM) use case diagram for the Prolan Block.

2.4.3 System Design

In System Design, the Software Designer builds the Platform Independent Model to define the software architecture, the interfaces between the components, and between the components and the overall software. To this end, (s)he uses structural diagrams, such as Component and Class Diagrams, and assigns the requirements to the system components. Since the viewpoint is platform independent, the interfaces are independent of any technological platform.

In the pilot project, Prolan used UML component and class diagrams to model the high-level architecture of the PB.

The PB design comprises five components (Figure 2.9): *Prolan Block Core Logic*, *Track Occupancy Detector*, *Network Communicator*, IS Controller, and *HMI Controller*.

By communicating with the axle counters, the *Track Occupancy Detector* notifies to the system events such as “a train entered the block” or “a train has left the block”. It also manages device failures, notifying exceptional conditions.

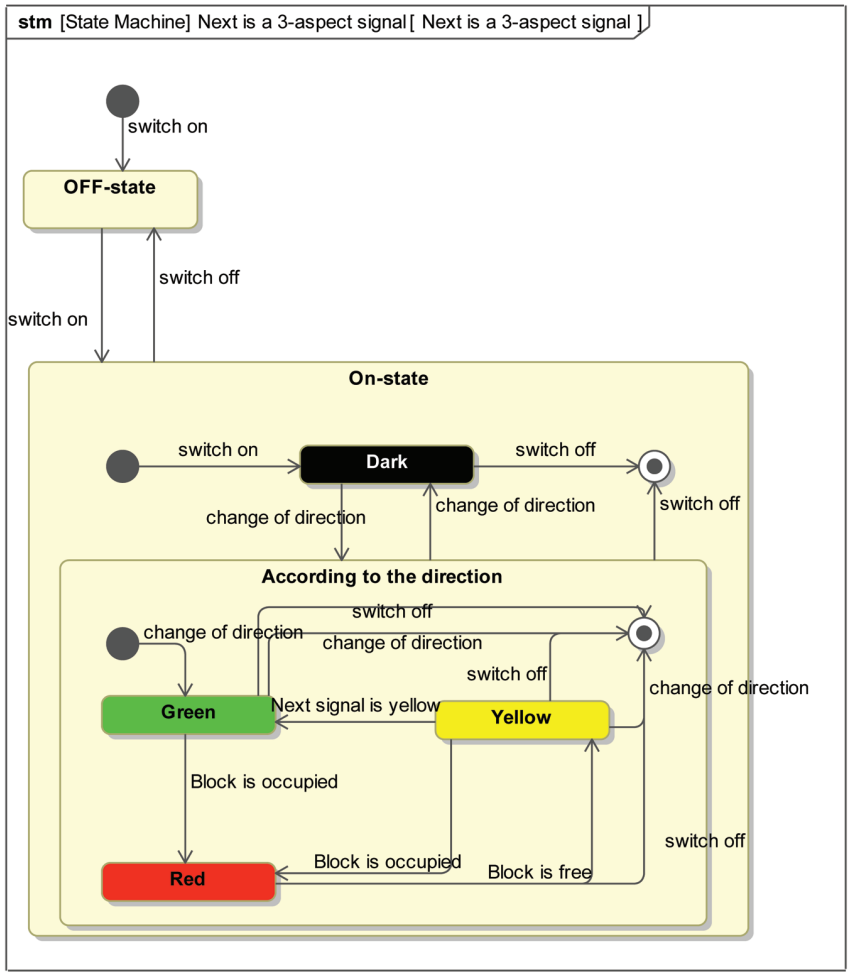

Figure 2.8 State machine diagram of the semaphore behavior [31].

The *IS Controller* interacts with the semaphore, setting the proper aspect and coping with malfunctionings. Similarly, the *NetworkCommunicator* uses the network, for interacting with adjacent PBs, and the *HMIController* manages the human-machine interface.

Finally, the *ProlanBlockCoreLogic* sets the interlocking systems according to its internal status and by collaborating with the other four components.

The components' interfaces were defined following guidelines to keep them as much as possible UML standard and platform-independent. For

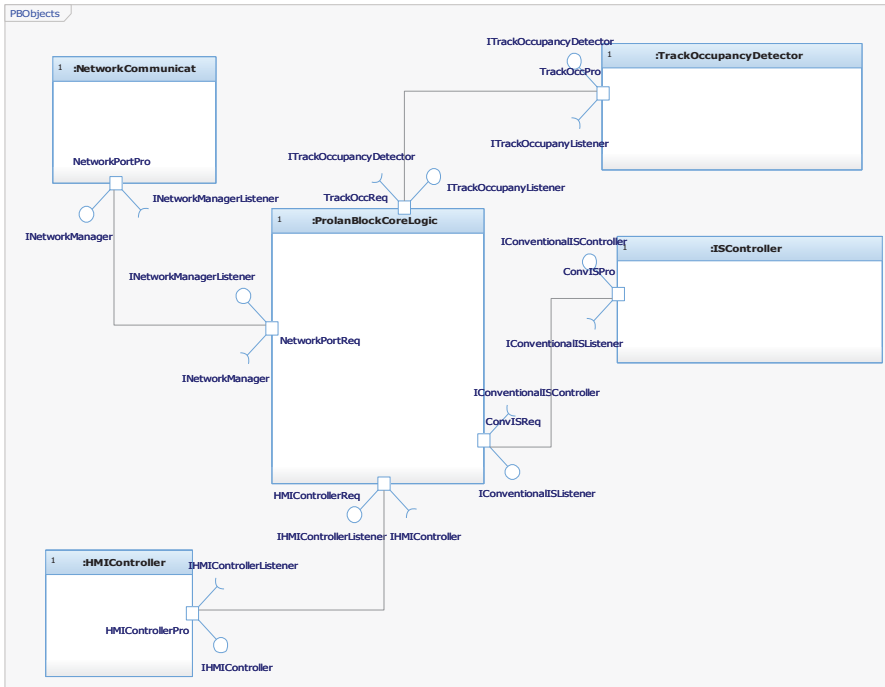

Figure 2.9 High-level system architecture [31].

instance the services of any middleware or library have been defined in terms of abstract interfaces, and the data types of the variables neglected one specific programming language.

2.4.4 Component Design

The next phase is Component Design: here the Designer refines the PIM, to specify the internal design of the components. (S)he identifies all lowest software units, fully detailing their input and output, and specifying algorithms and data structure. The PIM becomes complete, and can be runnable and object of simulation.

In the pilot project, Prolan defined the PIM using IBM Rhapsody Developer [33] (hereinafter: Rhapsody), but following the guidelines to build a model platform-independent. Indeed, the tool does not allow a clear separation between a PIM and a PSM, thus we avoided to insert C++ code, and adopted UML compliant syntax where possible.

Only few parts could not be specified in a platform-independent style. However, these parts were specified in C++, to exploit the model animation feature of Rhapsody. Indeed, Prolan was interested in this feature as a technique of early fault detection: Rhapsody's model animation generates an instrumented implementation of the model that allows to observe at runtime the program execution. This feature was useful and valuable for getting an immediate feedback on the design.

By Rhapsody Panel Diagrams, we drew a graphical user interface bound to the model that enabled to generate and receive model events at runtime. Of course, the execution can be also followed on behavioral diagrams, e.g., state machines or sequence diagrams.

2.4.4.1 Implementation

Implementation phase deals with the production of software that is analyzable, testable, verifiable and maintainable. Following MDA, the PIM is refined into one (or more) Platform Specific Models that are bound to target platforms. The PSM adds low-level implementation details. For instance, a PSM binds data and interfaces to the target OS and middleware chosen for the instantiation of the PIM.

Using IBM Rhapsody, Prolan set tagged values and other parameters to enrich the PIM with information platform-specific details, to specify how to translate the association (e.g., by static or dynamic arrays), what is the clock for the scheduler of the state machines' event queues, and other parameters. These are used by Rhapsody for the automatic translation of PSM into code.

Considering the requirements of the PB, Prolan specified that the generated code cannot use dynamic memory and that the variables have to be initialized at runtime, due to the lack of memory isolation on Prosigma, the technological platform for the PB. The platform specific code can also exploit the round-trip code feature of MDE tools:

1. By automatic code generation, packages, code skeletons, make files and other artifacts are automatically model-to-text produced;
2. The Implementer fills the code skeletons with platform-specific details, using the support of modern development environments (such as Eclipse);
3. By code round-trip, the model is automatically augmented with the information written manually by the Implementer in the source code.

According to the requirements, there are many options for the translator that include the execution framework at runtime. For instance, Rhapsody

offers two C/C++ frameworks: IBM Rhapsody Object Execution Framework (OXF), and IBM Rhapsody Simple Execution Framework (SXF). The latter is dedicated to embedded systems and safety-related development: qualification kits support the certification of the automatic generated code for several standard (including ISO 26262, EN 50128 and recently DO-178B).

The translation of our PSM in C++ source code generated around 7.5 thousands of lines of code for a platform using a conventional OS, 7.3 thousands for target platform using a commercial Real Time Operating System (VxWorks), and 5.9 thousands C lines of code for an embedded systems not using an OS. We used the SXF framework in the last code generation.

2.4.5 Validation Design

For Validation Design, we propose a model, named CIT Model, to specify the behavior of the actors and of the environment. CIT can be used for designing validation tests, e.g., as UTP sequence diagrams representing the interactions of the actors with the system.

Prolan decided to does not build a CIT model for the Prolan Block, and the benefits of CIT modeling have been experimented on another system, the Prolan Monitor, that is discussed in the next section.

2.4.6 Integration Verification Design

During Integration Verification Design, the Integrator realizes integration tests to show that components behave correctly when integrated together. The expected behavior of the components is independent from their inner design, thus we refer to this model is named Black Box Platform Independent Test Model (BB-PIT).

BB-PIT provides static and dynamic views of the system's components, and supports functional testing for unit/integration/system verification (Figure 2.10). The static description supports the generation of test harness, such as stubs and drivers for unit and integration testing. The dynamic description supports the generation of test suites and test cases.

The components' behavior seems modeled twice, in PIM and BB-PIT. However, the two models have different purposes: the first specifies how to build the system, and represents the specification that an actual implementation must comply with; the second describes the expected behavior in a way to verify its correspondence between requirements and implementation (e.g., by test cases).

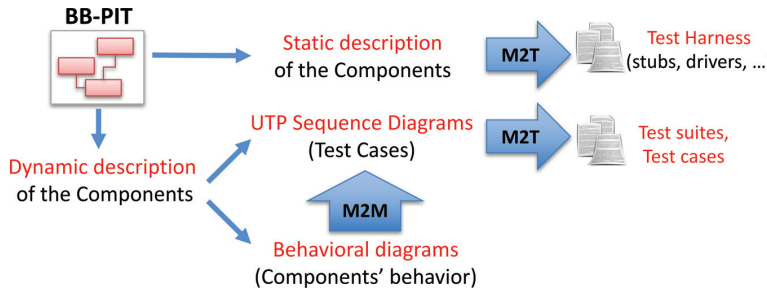

Figure 2.10 The transformations of the BB-PIT.

For tests specification, UML-UTP Sequence diagrams are less error-prone than textual notations, and it is easier to derive test cases for multiple target platforms (such as TTCN-3 and JUnit), enhancing reusability and maintainability.

In the pilot project, Prolan adopted Conformiq Designer (from here onward: Conformiq) [34] to generate automatically test cases from the BB-PIT. However, since Conformiq is not fully compliant with UML, the behavior was specified in QML, the language used by the tool. As adequacy criterion we used the requirement coverage, using the requirement traceability offered by the model-driven tools.

In total, Prolan achieved the full coverage of requirements generating 21 test cases for the *ProlanBlockCoreLogic*. Test cases were exported to JUnit from sequence diagrams (Figure 2.11), and the tool also provided us with the traceability matrix correlating test cases with the structural features they cover (states, transitions, requirements).

We also assessed the test harness generation from BB-PIT. Conformiq required to write the SUT adapter to let the testing framework interact with the system. Instead, Rhapsody offers the Test Conductor Add On [35], that automatically generates the testing harness (including the drivers and stubs), starting from model design diagrams. Within Test Conductor we could execute test cases directly in Rhapsody, observing the effects, and following the behavior of the SUT by means of sequence diagrams.

2.4.7 Component Verification Design

In Component Verification Design tests have to confirm that components perform their intended functions. Here, we define the Grey Box Platform Independent Test (GB-PIT) Model, which is used for verification by the internal view of the components. Following this flow, engineers

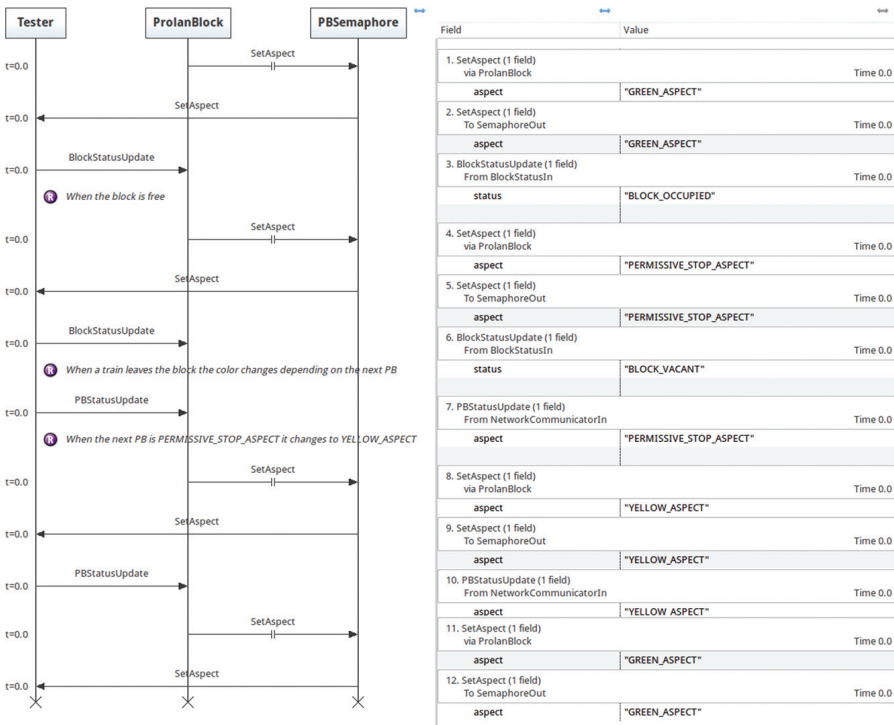

Figure 2.11 A test case automatically generated from the BB-PIT by Conformiq.

focus on a functional V&V modeling in the Integration Verification Design, whereas they focus on functional and structural V&V modeling at this stage.

Prolan assessed the IBM Rhapsody Automatic Test Generator (ATG) [36] for the structural verification of the *ProlanBlockCoreLogic*. ATG generated ten test cases (Figure 2.11) achieving the 91% coverage of the structural features of the model (they covered 19/21 states and 22/24 transitions). However, it was not able to reach the complete coverage.

2.4.8 Model-Driven V&V Subprocess

The activities on the right side of the V-Model concern V&V execution: we propose to use in these phases the models built during the Design activities, but after they have been refined with the new details added in PSM during implementation.

Thus, in Component Verification we named the model White Box Platform Specific Model (WB-PST), and the Tester adds new test cases considering the details of the target platform. The WB-PST can be used to calculate the test coverage on the basis of the final system code, as well as to support any kind of verification of the actual code, such as to derive consistent and efficient code review plans by considering the component software metrics and implementation details.

Similarly, during Integration Verification phase the BB-PIT model is refined in the Black Box Platform Specific Test Model (BB-PST), where platform specific details complete the integration test specification. For instance, the BB-PST can be exploited to perform interface testing, a technique that is highly recommended by CENELEC EN 50128: interface testing is executed knowing the actual domain of all interface variables, and selecting particular input to assess the behavior of the (integrated) components (e.g., at their normal, boundary, or invalid values).

Finally, in Validation phase, the Tester assesses that system and software requirements are met. To this end, (s)he executes the overall system tests defined in the CIT. Moreover, if the CIT is executable, the Tester can put the CIT and the software in-a-loop, to perform software-in-the-loop and hardware-in-the-loop testing.

2.5 Environment System Validation

Our methodology also exploits MDE for validation, by defining an environmental model during Validation Design to analyze the actors' behavior. This model, named CIT Model, specifies the expected behavior of the environment when interacting with the system by behavioral diagrams (e.g., Sequence, State Machine or Activity diagrams) that are used to derive validation test case.

The CIT also appears in other studies [37] but our definition differs from the previous ones, since we define the CIT a model for validation that abstracts from the computation details of the system under analysis (SUT), and also propose to develop the CIT as an executable model of the environment, with interfaces specular to those of the PIM. This definition supports multiple forms of V&V during the product life cycle.

Since the CIT has an interface specular to the one of the PIM, we can put the two models in-a-loop and the CIT can be used to perform model-in-the-loop test (since it is runnable), enabling to:

- Validate the system against its expected interactions with external actors;
- Create a simulated environment to reason about the operational aspects of the system in its environment (also through model animation).

Model-in-the-loop (MIL) testing can be performed as soon as the PIM is available, i.e., during the Component Design, enabling to an early fault detection. Moreover, if the Tester can use additional/external sources of knowledge to model the actors' behavior (such as domain knowledge or historical data), MIL testing can also be useful to detect missed software requirements, by assessing the behavior in a simulated environment.

Then, when a system implementation is available, the Tester can build an adapter to allow the CIT to interact with the actual SUT, allowing Software- and Hardware-in-the-loop Testing.

CIT also enables to performance testing, generally adopted for the assessment of critical systems, as it is recommended by the safety standards. Indeed, the CIT can generate inputs representative of the operational profile.

Other forms of verification allowed by the CIT are:

Model-checking through the in-the-loop model. This can assess the absence of undesired conditions during the operation, analyzing the states of the PIM and CIT.

Back-to-back testing, a special case is when the CIT can be seen as another PIM, for instance when we consider systems that act as client and server at the same time. For this kind of systems, we can instantiate two PIMs, putting in-a-loop each other, to perform back-to-back testing.

2.6 Experimenting the CIT

The benefits of the CIT and of environmental modeling have been assessed in another part of the interlocking system with which Prolan Block interacts, the Prolan Monitor.

The Prolan Monitor (PM) shares with the PB the Prosigma hardware and middleware platform, which is the basis of the next generation of Prolan's products.

The purpose of the PM is to send signals generated by legacy interlocking devices to modern interlocking systems that communicate through protocols based on IP networks (such as via X.25 over TCP/IP). More specifically, PM monitors *railway objects*: each object is associated to one bit of information, which is encoded by one couple of valent and antivalent physical signal values. The PM transmits the bit of information to other devices, detecting invalid values for the couple of electric signals. Indeed,

the input can suffer of special unstable states during which the signals quickly alternate in their value for a transient time, called *bounce time*: the PM must properly filter the signals, separating transient noise from invalid inputs.

To assess the benefits of the CIT, Prolan made an executable model of the PM's environment. The CIT is composed of two *CIT Railway Objects*: each *CIT Railway Object* controls the couple of logical signals associated with the binary information that they encapsulate; from the CIT point of view, the PM is an actor.

The *CIT Railway Objects* are implemented by a *Signal Generator* and an *Event Generator*: the *Event Generator* determines the next output to be triggered (including transient and invalid states), as specified by a user-defined operational profile, whereas the *Signal Generator* sets the couple of output signals and manages the duration of the transients.

A panel diagram makes the CIT interactive: a couple of knobs allow to set the event generation period and to customize the duration of transient states.

Linking together with an adapter simulating a physical relay the CIT and the PIM, we preliminarily performed Model-in-the-loop testing. Only changing the adapter with a real hardware card forwarding the events to the actual SUT, we could also perform Hardware in-the-loop testing (Figure 2.12).

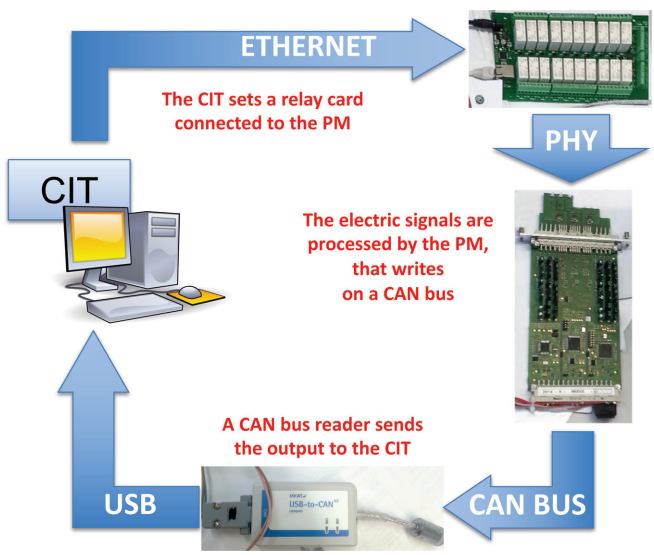

Figure 2.12 The configuration of the PM for HIL Testing.

2.7 Lesson Learned

The CECRIS knowledge transfer activities allowed assessing the maturity of MDE for railway interlocking systems. The project managers became acquainted with MDE methodologies and tools, and Prolan started to consider their introduction into the development processes.

Indeed, the pilot project showed that MDE is a mature technology, which supports the whole development process. Using SysML for requirements specification helped to produce better artifacts, reasoning formally on incongruences and missing specifications than with the current document-centric approach. Also, fast prototyping, early fault detection, automatic test generation, and other MDE features revealed a gain of productivity and quality during design and V&V phases than current methodology.

However, even for SME companies as Prolan that have limited engineering capacity, it is not easy to change current development process and practices. MDE requires for a technological and knowledge transfer. While the former can be addressed with personnel trainings, the latter is more subtle, long, and expensive. Therefore, Prolan submitted a joint tender proposal together with the Hungarian University, partner of CECRIS project, aiming at getting active support in their introduction during the next safety critical project, and to investigate model-driven technologies further.

Indeed, still an extensive experimentation of model-driven methodologies is needed. By the pilot project on the Prolan Block we qualitatively assessed MDE: even if the benefits of model-driven approaches turned out to be evident, we could not easily evaluate how much time is needed for Prolan to have a return of investment. This tender would introduce the academic know-how and support in the planning, design and implementation for a complete Railway Interlocking System project.

However, the current experience has been saved by Prolan, and if the tender is rejected then the enhancement of the current development lifecycle of Prolan with the one proposed in the framework of the pilot project will be applied. The innovation is planned to be applied gradually, through several stages expected to last several years.

At the first stage, Prolan targets to introduce models for supporting the current activities, starting to use MBE than MDE approaches. For instance, it is expected to adopt modelling tools for system requirement specification and system design phases. The current document-based system requirements specification and the the requirement management system will be replaced with software using SysML models, taking benefits of the improved model traceability.

These first changes already raise knowledge and technologies issues: we still have not decided if to adopt exclusively SysML for describing the system requirements, because we could be not able to teach satisfactorily modeling and SysML to all the team members; moreover, we have not completed the tool selection process. Among the selection criteria it is required that new tools be stable, and easily interoperable with the other software suites already in use at Prolan. The fully compliance with standards, and the vendor lock-in problem are also of interest for the tool selection. The relatively high price of licenses and the difficulty of using these modeling tools have an adverse effect on the introduction of model-driven methodologies.

The CECRIS experience revealed that model-driven technologies can really improve the development, enhancing quality of the product and of the development life cycle. Despite these advantages, the management of the railway product development decided to pursue a conservative approach towards the introduction of model based tools and development methods: the big issues of MDE concern skills and organization. The learning curve of these technologies is long and difficult to quantify, and the relevance of roles during the development will change, deeply impacting human-organizational factors. The innovation must be introduced gradually, taking into account these factors.

The benefits of cooperating with academic partners within the CECRIS project were manifold, as it was demonstrated by the pilot project: the experience in the methodologies and tools enabled Prolan to receive active tutoring and support for the full product lifecycle, shortening the learning curves and reducing the number of errors caused by the lack of knowledge, method, and experience with the tools. Moreover, the broad knowledge of the emerging new technologies in the field enabled the academic partners to suggest the criteria in selecting the appropriate tools and methodologies.

References

[1] Liebel, G., Marko, N., Tichy, M., Leitner, A., and Hansson J. (2014). "Assessing the State-of-Practice of Model-Based Engineering in the Embedded Systems Domain," in *Proceedings of the 7th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, eds. J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran (Berlin: Springer International Publishing), 166–182.

[2] Marko, N., Liebel, G., Sauter, D., Lodwich, A., Tichy, M., Leitner, A., and Hansson J. (2014). Model-based engineering for embedded systems

in practice. Research Reports in Software Engineering and Management, Technical report, University of Gothenburg, Gothenburg.

[3] Broy, M., Kirstan, S., Krcmar, H., Schätz, B., and Zimmermann, J. (2013). "What is the benefit of a model-based design of embedded software systems in the car industry" in *Emerging Technologies for the Evolution and Maintenance of Software Models* (Hershey, PA: IGI Global), 343–369.

[4] CECRIS. (2016). *EU Project CECRIS, CErtification of CRItical Systems*. Available at: http://www.cecris-project.eu

[5] Schmidt, D. C. (2006). "Guest Editor's Introduction: Model-Driven Engineering," in: *Computer 39.2*, 25–31. Lecture Notes in Computer Science (Berlin: Springer).

[6] Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. San Rafael, CA: Morgan & Claypool Publishers.

[7] Object Management Group (OMG). (2014). *MDA Guide* (Version 2.0). Available at: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01 (accessed on 2016-03).

[8] Baker, P., Dai, Z. R., Grabowski, J., Haugen, Ø., Schieferdecker, I., and Williams, C. (2007). *Model-Driven Testing: Using the UML Testing Profile* (New York, NY: Springer-Verlag New York, Inc.).

[9] Dai, Z. R. (2004). "Model-driven testing with UML 2.0," in *Proceedings of the 2nd European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA)*, eds D. Akehurst, 179–187. Tech. rep. 17-04, University of Kent, Canterbury.

[10] Kent, S. (2002). "Model Driven Engineering," in *the Proceedings of the Third International Conference on Integrated Formal Methods (IFM)*, 286–298. Berlin: Springer-Verlag.

[11] Object Management Group (OMG). (2003). *MDA Guide (Version 1.0.1)*. Available at: http://www.omg.org/cgi-bin/doc?omg/03-06-01 (accessed on 2016-03).

[12] Davies, I., Green, P., Rosemann, M., Indulska, M., and Gallo, S. (2006). How do practitioners use conceptual modeling in practice? *Data Knowl. Eng.* 58.3, 358–380.

[13] Forward, A. and Lethbridge, T. C. (2008). "Problems and Opportunities for Model-centric Versus Code-centric Software Development: A Survey of Software Professionals," in *Proc. of the International Workshop on Models in Software Engineering (MISE)* (New York, NY: ACM), 27–32.

[14] Hutchinson, J., Rouncefield, M., and Whittle, J. (2011). "Model-driven engineering practices in industry," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)* (New York, NY: IEEE), 633–642.

[15] Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011). "Empirical Assessment of MDE in Industry," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)* (New York, NY: ACM), 471–480.

[16] Hutchinson, J., Whittle, J., and Rouncefield, M. (2014). "Model-driven engineering practices in industry: social, organizational and managerial factors that lead to success or failure," in *Science of Computer Programming 89, Part B* (Amsterdam: Elsevier), 144–161.

[17] Whittle, J., Hutchinson, J., and Rouncefield, M. (2014). "The state of practice in model-driven engineering," in *IEEE Software 31.3* (New York, NY: IEEE), 79–85.

[18] Tomassetti, F., Torchiano, M., Tiso, A., Ricca, F., and Reggio, G. (2012). "Maturity of software modelling and model driven engineering: A survey in the Italian industry," in *Proceedings of the 16th International Conference on Evaluation Assessment in Software Engineering (EASE)* (New York, NY: ACM), pp. 91–100.

[19] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., and Reggio, G. (2011). "Preliminary Findings from a Survey on the MD State of the Practice," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)* (New York, NY: ACM), 372–375.

[20] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., and Reggio, G. (2012). "Benefits from modelling and MDD adoption: expectations and achievements," in *Proceedings of the 2nd International Workshop on Experiences and Empirical Studies in Software Modelling (EESSMod)* (New York, NY: ACM), 1–6.

[21] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., and Reggio, G. (2013). Relevance, benefits, and problems of software modelling and model driven techniques – A survey in the Italian industry. *J. Syst. Softw*. 86.8, 2110–2126.

[22] Agner, L. T. W., Soares, I. W., Stadzisz, P. C., and Simo, J. M. (2013). A Brazilian survey on UML and model-driven practices for embedded software development. *J. Syst. Softw*. 86.4, 997–1005.

[23] Mussbacher, G., Amyot, D., Breu, R., Bruel, J. M., Cheng, B. H. C., Collet, P., Combemale, B., France, R. B., Heldal, R., Hill, J.,

Kienzle, J., Schöttle, M., Steimann, F., Stikkolorum, D., and Whittle, J. (2014). "The relevance of model-driven engineering thirty years from now," in *Proceedings of the 17th International Conference on Model-Driven Engineering Languages and Systems (MODELS)* eds. J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran (New York, NY: Springer International Publishing), 183–200.

[24] Mohagheghi, P. and Dehlen, V. (2008). "Where Is the Proof? A Review of Experiences from Applying MDE in Industry," in *Proceedings of 4th European Conference on the Model Driven Architecture – Foundations and Applications (ECMDA-FA)*, Vol. 5095, ed. I. Schiefer-decker and A. Hartman. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 432–443.

[25] Baker, P., Loh, S., and Weil, F. (2005). "Model-Driven Engineering in a Large Industrial Context – Motorola Case Study," in *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, eds. L. Briand and C. Williams (Berlin: Springer), 476–491.

[26] Weigert, T. and Weil, F. (2006). "Practical experiences in using model-driven engineering to develop trustworthy computing systems," in *Proc. of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, Vol. 1 (New York: IEEE), 208–215.

[27] Huhn, M. and Hungar, H. (2010). "8 UML for software safety and certification," in *Model-Based Engineering of Embedded Real-Time Systems: International Dagstuhl Workshop. Revised Selected Papers*, eds. H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schätz (Berlin: Springer), 201–237.

[28] Pettit, R., Mezcciani, N., and Fant, J. (2014). "On the needs and challenges of model-based engineering for spaceflight software systems," in *Proceedings of the IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)* (New York: IEEE), 25–31.

[29] Ferrari, A., Fantechi, A., and Gnesi, S. (2012). "Lessons learnt from the adoption of formal model-based development," in *Proc. of 4th International Symposium on the NASA Formal Methods (NFM)*, eds. A. E. Goodloe and S. Person (Berlin: Springer), 24–38.

[30] Object Management Group (OMG). (2008). Systems modeling language (SysML). Available at: http://www.omg.org/docs/formal/08-11-02.pdf (accessed on 2016-03).

[31] Scippacercola, F., Pietrantuono, R., Russo, S., Zentai, A. (2015). "Model-driven engineering of a railway interlocking system," in *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2015)* (Setúbal: SCITEPRESS), 509–519.

[32] No Magic, Inc. (2016). *Magic Draw. MagicDraw*. Available at: http://www.nomagic.com/products/magic-draw.html (accessed on 2016-03).

[33] IBM Corp. (2016). *Rational® Rhapsody® Developer*. Available at: http://www-03.ibm.com/software/products/it/ratirhap (accessed on 2016-03).

[34] Conformiq Inc. Conformiq Designer. http://www.conformiq.com/products/conformiq-designer, (accessed on 2016-03).

[35] IBM Corp. (2016). *Rational® Rhapsody® Test Conductor Add On*. User Guide. Available at: http://pic.dhe.ibm.com/infocenter/rhaphlp/v7r6/topic/com.ibm.rhp.oem.pdf.doc/pdf/RTCUserGuide.pdf (accessed on 2016-03).

[36] IBM Corp. (2016). *Rational® Rhapsody® Automatic Test Generator Add On*. User Guide. Available at: http://pic.dhe.ibm.com/infocenter/rhaphlp/v7r5/topic/com.ibm.rhapsody.oem.pdf.doc/pdf/ATG-UserGuide.pdf (accessed on 2016-03).

[37] Schieferdecker, I. (2005). "The UML 2.0 Test Profile as a Basis for Integrated System and Test Development," in *Proceedings of Köllen Druck+Verlag GmbH, Jahrestagung der Gesellschaft für Informatik* (Germany: Köllen Druck & Verlag GmbH), Vol. 35, pp. 395–399.