Timing Analysis using the MARTE Profile in the Design of Rail Automation Systems

M Hagner, M Huhn, A Zechner

To cite this version:

HAL Id: hal-02270283
https://hal.archives-ouvertes.fr/hal-02270283

Submitted on 24 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract: For dependable systems as in the railway domain the timing behaviour is considered part of the functional correctness. Thus timing requirements have to be traced and refined through the system and software development phases and validation and verification efforts have to address the timing as well as the pure input/output behaviour.

We show how timing can be handled in a UML or SysML based approach to the development of software-intensive railway systems by using the new MARTE profile. Thereby timing becomes fully integrated in the chain of system and software models and may benefit from tool support. Moreover, automated timing analysis may be employed via model transformations which enables the exploration of timing-related issues in various design phases.

Keywords: MARTE, Timing Analysis, Rail Automation Systems, CENELEC

1. Introduction

Dependable systems are in many cases hard real-time systems i.e. if the timing behaviour violates the requirements, the system reaction has to be considered as faulty because of potential severe consequences. Therefore, timing requirements as other runtime properties, which are in non-dependable software design considered as extra-functional like availability, are taken into account from the very beginning of the development process. Within each design phase a catalogue of measures is employed to assure predictable behaviour.

However, nowadays standardised tool-supported model driven development for the application logic on the basis of UML or SysML is on the rise in the railway domain. Moreover, today’s rail automation systems are composed out of a set of configurable components to improve reuse. On the other hand, COTS products for hardware components and lower middleware layers find their way into this segment dependable systems. All these require a new treatment of real-time requirements that supports tracing through the model driven design and automated timing analysis for verification. An answer is MARTE (A UML Profile for Modelling and Analysis of Real-Time and Embedded systems) [5], a new UML profile extension for embedded and real-time systems. It is proposed by the “Pro-Marte” consortium consisting of academics, tool providers, and OMG end-users. The goal of the consortium was to define a concept, in terms of UML extensions, to model real-time and embedded systems. The MARTE Profile is a successor of the Profile for Schedulability, Performance, and Time (SPT Profile) [6] and the QoS Profile (Profile for modeling quality of service and fault tolerance characteristics and mechanisms) [7].
CENELEC standards EN50126, EN50128 and EN50129 give recommendations to railway companies, industry and suppliers for consequently applying quality management (RAMS\(^2\)). Application of those standards has to be carried out systematically during all phases of a product’s life cycle.

### 2.1 Quality Management

Due to the fact that failure in safety relevant functions of railway systems may have dramatic consequences, the main focus in RAMS management lies on the safety aspect. CENELEC standards call for an assurance that the risk of severe hazards is minimized to an acceptable risk \(^8\) which is usually expressed as Tolerable Hazard Rate (THR). System failures may occur from two different kinds of reasons:

- random or aging effects on the material
- systematic errors from incorrect design

For the first probabilistic methods can be applied, but the second kind needs another treatment since systematic errors cannot be quantified with mathematical means. In order to prevent systematic errors and random failures, the concept of Safety Integrity Level (SIL) is applied. SIL is a classification related to ranges of THRs. As a SIL comes along with a catalogue of measures to be applied in the different phases of a product’s life cycle, SIL addresses in particular systematic errors in the design by recommending methods and practices for the development process.

### 2.2 Development Processes

According to EN50126 \(^1\) the system development process comprises several phases: concept, system definition, risk analysis, system requirements, apportionment of system requirements, design, and implementation. Further refinement of the process leads to the loosely coupled development processes for hardware and software. When designing and assembling hardware made of COTS and rather simple electric circuitry most of the RAMS qualities can be addressed directly because of the physical and probabilistic properties of the single parts from resistors to complete electric circuits. Software is immaterial and does not suffer from statistic errors that is why EN50128 \(^9\) introduces Software SIL. As a consequence the standard makes recommendations for tracing requirements, using formal or semi-formal methods for design and verification and execution of certain test procedures according to the SSIL that is assigned to the function or subcomponent under development. Furthermore EN50128 proposes phases and artefacts for a software development process (see Figure 2).

The development process for software starts with the software requirements specification phase. As inputs serve documents from system requirements specification, system safety requirements, system architecture specification, and software quality assurance plan; outputs are requirements and test requirements. Recommendations for this phase include formal, semi formal, and structured methods like CCS, temporal logic, function block and sequence diagrams, SDL, Yourdan, etc.

Software requirements specification, system safety requirements, system architecture specification, and software quality assurance plan form the basis of the software architecture phase. The software architecture specification has to identify all software components and subsystems, document and analyse interactions between software and hardware, and provide traceability of requirements. The standard proposes fault tree analysis, software error effect analysis, and several programming techniques and best practice architecture fragments.

A design and implementation phase basically ends the constructive part of the software development process. The design has to give detailed information on interfaces of all software components and modules. Every kind of module has to be specified exactly regarding data structures, diagrams of data and control flows, and algorithms. Software module specifications shall be sufficiently detailed for further implementation and coding. Here again recommendations for formal, semi formal, and structured methods can be applied. Object-oriented or module-oriented approaches, coding guidelines, programming languages with certain properties, restriction to a subset of a programming language, utilisation of approved or certified software libraries, and other programming related issues extend the list of recommendations.

Model checking and the like are methods for proving pure functional correctness in the meaning of reactive, computational behaviour. Besides timed Petri nets or temporal logic the CENELEC standards give only marginal recommendations for checking, tracing, and proving correct temporal behaviour. In particular this applies to the architecture and design phases. However, this does not reflect the state of the art in real-time modeling and analysis and in particular for complex dependable systems it is mandatory to assure timing correctness as that timing is a safety relevant matter whether for the functional behaviour or for the prevention of hazards to minimize detection time of failures and to initiate fail safe reactions.

---

\(^2\)European Committee for Electrotechnical Standardization

\(^3\)Reliability, Availability, Maintainability, and Safety

\(^4\)Components Off The Shelf
3. Case Study

The German railroad network comprises about 50,000 level crossings. According to the laws and regulations of Germany and Switzerland railway lines may be equipped with crossings where trains travel with up to 160 km/h. Due to this European railway standards [1] demand high requirements for automatic protection systems concerning reliability, availability, maintainability and safety.

In this case study we will examine the development of an automatic train level crossing system with supervision signal. The purpose of such a system is to prevent street traffic from entering the crossing while trains are approaching it or pass through the crossing. A level crossing system with supervision signal works as an autonomous system; interaction with a railway control centre or interlocking is not necessary.

3.1 Basic Execution of Protection

When railway vehicles are approaching a level crossing, road users have to be signalled. On technically safeguarded level crossings this is realised indirectly via the train activating the protection process. The interval of time between announcing the approaching train to its arrival on the crossing must be sufficiently long for road users to either come to a halt before the rails or to enter the area and to safely clear before the train arrives at the level crossing. Not until successful safeguarding the train may pass the level crossing. When the complete train has depleted the level crossing, road users may be given way to cross the rails.

3.2 System Boundaries

A protection system for level crossings obviously has boundaries towards its environment: the railway system (train, rails, etc.) and road traffic. Laws and regulations prescribe more detailed restrictions on the concrete interfaces. The level crossing protection system has to warn road traffic by a set of lights comprised of a yellow and a red light with conventional meaning. Additionally the level crossing can also have gates faced toward the street. A railway signal, namely the supervision signal, shall inform the train driver of the state of the level crossing. Besides those prescribed interfaces a protection system also needs an interface component for detecting trains.

Regarding the fact that we will focus on software development - ignoring the true physical boundaries to the environment like climate conditions, electrical interfaces, etc. - our list of interfaces is sufficient.

3.3 Variants of Automatic Level Crossing Protection

Protection systems for level crossing are conceived for different operational purposes. Besides the design with supervision signal other systems exist which are remotely controlled by interlockings from railway control centres or manually operated from station personnel. As initially mentioned a protection system with supervision signal works independently from other train operating or control systems which could influence train movement and prevent a train from entering when the crossing is not guarded against road traffic. In this variant the maximum speed of trains is limited to 120 km/h. The supervision signal is faced towards approaching trains and shows whether the protection system works and the area can be entered safely or not. Its placement is in a safe stopping distance from the grade crossing; so the train driver can react and initiate braking. The signal can show two signal aspects LC1, a blinking white light indicating that the facility is safe, or LC0 otherwise. Its activation takes place as soon as the set of lights is showing red.

3.4 Parameters and Constraints

Regarding the operation of protection described earlier we get several constraints and parameters which have to be taken into account when developing a protection system:

- Maximum allowed speed of trains
- Diameter of level crossing
- Maximum expected length of road vehicles
- Minimum expected depletion time of road vehicles
- Stopping distance of road vehicles including time to react
- Train deceleration at emergency brake
- Reaction time of the train driver

Most of those parameters and constraints result in requirements as we will see later.

4. The Development Process

Here we show how to integrate the MARTE profile for specifying real-time properties in a model based development process conforming to CENELEC. Our focus lies on software development but MARTE can be used from early phases of requirement specification on the system level till implementation. We will go through all development phases but after the system architecture phase we concentrate on software development5 and will introduce artefacts of the hardware parts only when required for understanding.

4.1 Phase 0 - User Requirements Specification

In the railway domain user requirements are stated by railway companies, which typically have a long history. Alike is the form of user requirements in this domain. Some come from descriptions of operating procedures for railways - mostly written for railway personnel -, sometimes involving descriptions of existing technology. The operation procedures contain explicit time constants for end-to-end paths for functions such as safeguarding the level crossing. Furthermore legal constraints also have to be considered as a source of requirements, next to the tacit knowledge of domain experts. However, in this paper we refer to the more or less formal description of the case study as the user requirements specification in our development process.

5HW development will ideally take place in parallel.
4.2 Phase 1 - System Requirements Specification and System Architecture

Unlike all other phases this phase is twofold, it contains all system level activities of the development reflecting our concentration on software development. System requirements capture the user requirements and enrich them with goals and requirements of the developing manufacturer like for instance product line development. Furthermore requirements are usually conditioned for further engineering use thus they are getting more technical.

Thereafter follows system architecture. Major system suppliers for railways typically have existing solutions and platforms which fulfill typical requirements of railway standards; especially some generic documents for certification purposes already exist. A system architecture specification collects solution ideas to address functional and extra-functional requirements making use of generic components, COTS, and deciding where new subsystems either hard- or software have to be designed. Using MARTE can help to improve the architectural decision process by making inherited and derived requirements more explicit.

4.2.1 System Requirements Specification

The level crossing system consists of gates, a set of lights, a supervision signal, and activation and deactivation sensors to detect approaching or leaving trains. The arrangement of these elements is depicted in Figure 3.

![Figure 3: Schematic view on the level crossing system](image)

If a train approaches to pass a level crossing, it first passes the activation sensor (e.g. an axle counter). Hence, the level crossing system recognizes a train coming and starts to prevent the road traffic from crossing the rails. Therefore, it switches the traffic lights yellow. It is well-defined, how long the lights have to stay yellow. If the level crossing is constructed for traffic speeds up to 70 km/h, the set of lights have to stay yellow for three to five seconds. Next the lights are switched to red. The level crossing system has to monitor whether turning the red light on was successful, i.e. the status from the set of lights has to be read. Only afterwards the supervision signal is switched on (showing signal aspect LC1). About 7 seconds later the gates start to close. The time required for closing the gates depends on the length of the gate’s bar. If the bars are longer then 6 meters, it can take up to 10 seconds and if they are shorter, 6 seconds.

Right beyond the level crossing is the deactivation sensor. This sensor detects the leaving train; following the gates can be opened, the traffic lights are turned off, and the supervision signal shows LC0 again.

One requirement of the level crossing system is that the actions to protect the level crossing have to be executed fast enough. The supervision signal has to be in LC1 mode 7 seconds before the train passes the signal. These 7 seconds are necessary for the engine driver to recognize the signal and are defined in the engineering standards. If the supervision signal is not in LC1 mode, the engine driver has to stop the train.

The time that remains to switch the supervision signal to LC1 depends on the distance between the activation sensor and the supervision signal. If the distance is longer, the train will take longer to reach the point where the driver has to recognize the supervision signal, thus the system has more time for its interactions. Because of cable costs and costs for the installation and maintainance it is an effort to keep the distance between the sensor and the supervision signal as short as possible. However, the switching order and time intervals inbetween, a minimal distance between the activation sensor and the supervision signal can also be calculated. The distance between the supervision signal and the level crossing is fixed, because it directly depends on the braking distance of the train at maximum speed. In our case study the distance between the activation sensor and the supervision signal is defined to 450 m, which implies that the supervision signal should be in LC1 mode 6.5 seconds after the train passes the sensor (7 seconds before the train passes the supervision signal). Then, the red traffic light is on and 7 seconds later, the closing of the gates should begin. It takes up to 6 seconds afterwards to close the gates. As a result of these constraints, the complete protection of the level crossing has to be finished after 20 seconds (time to switch the supervision signal + red light phase + time to close the gates + processing and communication time). This is one of the main requirements.

4.2.2 System Architecture

In this phase, there is no detailed resource modelling. A fragmentary choice about the hardware architecture is done, mainly influenced by experiences made in previous projects or by applicable COTS. However, there is no final decision about e.g. a specific CPU. Tasks are not described in this phase, because tasks are not yet identified. Hence, no detailed scheduling analysis is possible. A more considerable method at this point of the development is to refine and capture the requirements in the UML model (like execution path deadlines of functionalities etc.) and define the interfaces (which components are connected with each other). Therefore, UML diagrams are extended using the MARTE profile.

Figure 4 shows a component diagram of the level crossing system. The `LevelCrossingSystem` offers two services (represented as methods): one service to prevent the traffic...
from crossing the rails and switch the supervision signal in LC1 mode (\texttt{protectLevelCrossing}) and one service to open the level crossing again for the traffic and to switch the supervision signal to LC0 mode (\texttt{unprotectLevelCrossing}). These two services represent the main functionality of the system. The train component represents an approaching train that triggers the protection of the level crossing (the service \texttt{protectLevelCrossing}). Afterwards the level crossing system sets the lights, switches the supervision signal, and closes the gates. The execution flow of these actions is depicted in Figure 5. There is a similar diagram for the \texttt{unprotectLevelCrossing} service of the level crossing system.

The stereotypes used for this diagram are defined in the MARTE specification for a MARTE Design Model:

\texttt{rtUnit} - "rtUnit" marks the component as a real-time unit.
\texttt{rtService} - This stereotype implies that this function is a real-time service.
\texttt{rtf} - This stereotype gives the developer the possibility to use the tagged value "absDl" and "occKind".

In Figure 4 the \texttt{LevelCrossingSystem} is represented as a component using the stereotype "rtUnit". The methods of this component (the services) are marked with the stereotypes "rtService" and "rtf". With the "absDl" and "occKind" tagged values a deadline and the arrival pattern of the initiating event can be defined for the corresponding functionality. The values are directly derived from the requirement specification. One requirement is that the protection of the level crossing has to be finished after 20 seconds. This is now captured in the component diagram with the deadline of the \texttt{protectLevelCrossing} service.

A refinement of the requirements is already done in this phase. The user requirements specification defines that the \texttt{protectLevelCrossing} service has a deadline of 20 seconds. Another requirement is that the supervision signal has to be in LC1 mode after 6.5 seconds. The following equation shows, how the requirements are distributed:

\[ 6.5\, s = t_{\text{yellow}} + t_{\text{communication}} + t_{\text{processing}} \]

Derived from maximum car length, stopping distance of road vehicles, and their minimum depletion speed of the crossing, follows that the lights have to be yellow for 5 seconds (\( t_{\text{yellow}} = 5\, s \)). Hence, it is easy to calculate that there are 1.5 second left for the communication between the elements of the level crossing (\( t_{\text{communication}} \)) and for the processing of the commands (\( t_{\text{processing}} \)). This is possible for this case study, because of the low complexity of the interactions of the level crossing system (i.e. there are no backward loops or data dependencies thus simple addition of numbers or intervals suffices).

The execution flow of the actions can be defined in sequence diagrams or activity charts. Figure 5 represents the actions of the safeguarding process. In this diagram, it is possible to give a more detailed description about runtime of actions or requirements. Therefore, the Value Specification Language (VSL) can be used. In Figure 5, two variables are defined. The first \( t_0 \) is the mark, where the service \texttt{levelCrossingSystem} is triggered. Later the set of lights should switch to red. Afterwards the supervision signal should be set to LC0 (the light blinks soon enough for the engine driver that the train does not need to be stopped). This has to happen 6.5 seconds after the approaching train triggered the level crossing system. From 12 to 14 seconds after the train triggered the level crossing system, the system starts to close the gates. The last requirement for this service is that the gates have to close within 6 seconds. Again, all values are directly derived from the requirements specification.

4.3 Phase 2 - Software Requirements Specification

At this stage of development as a result of the system architecture phase apportionment of functions and assignment to COTS, pure hardware and software has already been accomplished on a level of coarse subsystems. Requirements regarding functions assigned to software have to be collected and refined. In an iterative execution of the process the boundaries of the software system will get clearer over time due to results and choices from the concurrent hardware architecture process. Thus, high-level functions assigned to software realization have to be split to meet the new boundaries.

In our case study, functional integration and control of all hardware and COTS systems shall be performed in software. The software has to read and interpret values from axle counters and simple I/O- hardware. On the other hand it controls and sends data in given telegram message formats to those elements. Some parameters e.g. bar length type and a lot more have to be configurable.
Although software or software architecture itself cannot fulfill timing requirements on its own it is inevitable to inherit and track such requirements for later phases. Nevertheless, architectural decisions on the software level may influence satisfiability of requirements positively, e.g. in separating timing critical from other functions [10], or negatively. In addition choosing architectural styles or best practice methods and solutions for the software infrastructure, can ease predictive validation of timing requirements.

4.4 Phase 3 - Software Architecture and Design

As in this phase the software structure is developed, the architecture and the behaviour of the system is described in more detail. Hence, it is possible to give more precise predictions about the timing requirements of the system. These predictions can be made using scheduling analysis tools (e.g. SymTA/S [2] or MAST [4]). The metamodels of (UML or SysML based) CASE tools and timing analysis tools differ and thus, an analysis on the basis of UML models is only possible after a transformation from the UML metamodel into the metamodel of the chosen scheduling analysis tool. Moreover, the UML model has to be completed with respect to timing analysis: It is necessary to add priorities, scheduling algorithms, task execution times etc. To keep this information added for timing exploration separated from requirements and design decisions, one possibility is to create a scheduling analysis view [11]. This view should contain all necessary information for a scheduling analysis.

The scheduling analysis view uses the following stereotypes:

**SaExecStep** - A “SaExecStep” is a kind of step that represents a usage of a resource, like a CPU.

**SaCommStep** - A “SaCommStep” is a kind of step that represents a usage of a communication media.

**SaEnd2EndFlow** - End-to-end flows describe a unit of processing work in the analyzed system, which contend for use of the processing resources.

**GaWorkloadEvent** - Defines a stream of events that make up a workload which drives the system.

**allocated** - This stereotype maps a “SchedulableResource” on a “SaExecHost” or “SaCommHost”.

**SharedResource** - Resources like memory or I/O devices are described with the “SharedResource” stereotype.

Most of the introduced stereotypes are defined in the MARTE specification of a scheduling analysis model (MARTE Analysis Model). Our collection of stereotypes is based on the metamodel of scheduling analysis tools like SymTA/S.

Figure 6 shows a class diagram of the scheduling analysis view that describes the structure of the level crossing system. The functionalities/the tasks are represented by methods of the “SchedulableResource” marked classes. The “SchedulableResources” are mapped on resources (like processors or busses). This is done using the “allocate” stereotype. The tasks are described using the “SaExecStep” stereotype. All components communicate with each other over a bus. The methods that represent the communication are extended with the “SaCommStep” stereotype.

Figure 6 depicts the entire level crossing system. Beside other things, it consists of a component **ControlUnit**. This component is refined in the class diagram presented in Figure 7 which lists all tasks of the **ControlUnit**. Part of this list are tasks to check the behaviour of the unit (systemIntegrityCheck, cycleTimeMonitoring), tasks to control the set of lights (sendSetYellow, checkRedStatus . . .), tasks to initiate the closing and opening of the gates (closeGates, openGates), etc.

Details about tasks and scheduling parameters are examined during this phase. These details can now be added to the model using tagged values (see Figure 8). Thus, it is possible to define deadlines, execution times, priorities, etc.. For
a scheduling analysis, it is necessary that all time/scheduling concerning information are added to the model. The execution times can be estimated from experts or measured from previous projects. The response times are calculated using the analysis tools, and afterwards transferred back into the model (where the variable r1 is defined in Figure 8).

The dependencies of tasks and the execution order can be illustrated using activity or sequence diagrams. The diagrams are used to show the workload behaviours. This is done using the “GaWorkloadEvent” and the “SaEnd2EndFlow” stereotypes. The corresponding tagged values offer enough opportunities to describe the situations (e.g. the arrival pattern of the approaching train or the deadline of an execution path).

One important requirement for the level crossing system is that the supervision signal is in LC1 mode soon enough for the engine driver to recognize it, so the train does not need to be stopped. The order of the actions, that are executed when a train approaches, has been described in the system requirements specification (section 4.2). This workload situation is described using sequence or activity diagrams.

It is possible and one of the main purposes of the scheduling analysis view, to transform a model into the format of a scheduling analysis tool. The analysis results show whether deadlines are met, execution paths are executed fast enough, etc. The results of the analysis can be transferred back into the UML model. In Figure 9 a part of the level crossing system is depicted in SymTA/S. The tasks are mapped on resources and connected using event streams. On the SymTA/S model an analysis can be executed automatically.

The hardware is developed in parallel and so the scheduling analysis view may change iteratively. New hardware decisions may have great impact on the timing behaviour of the system, because of their influence on execution times, blocking times etc. For the hardware development, the MARTE profile can also be used. Elements for detailed hardware modeling are part of the MARTE design model. Nevertheless, the focus of this document lies on the software modelling. The analysis results of the scheduling analysis view may also have influence on the hardware development. If a result of the analysis is that a utilization of a resource is too high or a deadline is missed, it may be a solution to employ faster hardware.
4.5 Phase 4 - Model-Coding

In this phase, the design, implementation, and calibration is done. It is still recommended to use scheduling analysis during the implementation and calibration phase. As the structure of the system developed in phase 3 is fixed, the model (especially the scheduling analysis view) can still be used for further analyses. The task execution times now can either be calculated or measured times. Therefore tools like aiT\(^6\) or RTA Trace\(^7\). As now code exists the realised hardware/software system can be validated for the first time. However, using formally founded worst case execution time calculations like done by aiT and scheduling analysis is of additional value, because then the analysis result is a verification and not only a simulation based and thus incomplete statement.

5. Lessons Learned

MARTE offers manifold possibilities for modeling embedded systems. Because of the broad scope of the profile, it is non-trivial to select an appropriate set of elements for a specific modeling purpose that also meets to the needs of a dependable system design process as a whole like seamless tracing or completeness for automated analysis. This is true even more, because in the manufacturing of complex software systems many developers only have a partial view on the design. In discussions with practitioners it turned out that a restricted set of MARTE profile elements and guidelines for their use in a certain development phase is preliminary for a successful transfer into practice. Modeling rules and guidelines relieve the developer from the complexity of UML and the MARTE profile and therefore increase productivity because the developer can concentrate on the modeling but integrate the timing easily. As a step in that direction we defined a scheduling analysis view\(^11\) with guidelines to assure the extension of a design with timing and resource information leads to a model that is complete w.r.t. a scheduling analysis. Moreover, the timing related information, that is exclusively required for analysis purposes, is added in a separated view to the model to avoid the overloading of the design model in particular with temporary information that is only of use to explore and evaluate a number of design alternatives.

A major benefit of modeling guidelines is that transformations to analysis tools can be easier automated. This is due to the fact that the automatic transformations can be kept simpler if the structure of the MARTE extended UML model is restricted. This is especially helpful for the scheduling analysis views, used in phase 3 (see section 4.4).

On the OMG MARTE homepage\(^7\) the Papyrus for UML tool\(^8\) is proposed. The Papyrus for UML project offers an add-on for their modeling tool that includes all MARTE elements. With this tool and the add-on, it is possible to create models conforming to MARTE. Papyrus for UML is still under development. Thus, there are some features, which do not work properly (e.g. using

---

\(^6\)http://www.etas.com/de/products/rta_trace.php
\(^7\)http://www.omgmar.te.org
\(^8\)http://www.papyrusuml.org
activity charts) until now.

6. Conclusion

We have presented how to use the MARTE profile within system and software development process conforming to domain specific standards for safety critical systems. In the early phases, it helps to capture and visualise the timing requirements with UML conforming notations. In the later phases, MARTE annotations can be used for scheduling analysis before any implementation takes place. As a case study we have shown how to enrich a model of a level crossing protection system for scheduling analysis. The extended model can be used as an input for automatic scheduling analysis tools. The CENELEC standards give only marginal recommendations for checking, tracing and proving correct temporal behaviour. As we have shown, the integration of timing requirements into a model based, tool supported design methodology is useful and necessary. The future versions of the standards should be extended by recommendations for acquisition and refinement of timing requirements during the development of safety critical and programmable systems.

7. Acknowledgements

We are grateful to Stefan Gerken from Siemens Transprotation Systems for fruitful discussions on a subprofile applicable in practice as well as the Symtavision GmbH for the grant of free software licenses.

8. References


