openETCS/WP2/D2.1

ITEA2 Project
Call 6 11025
2012 – 2015

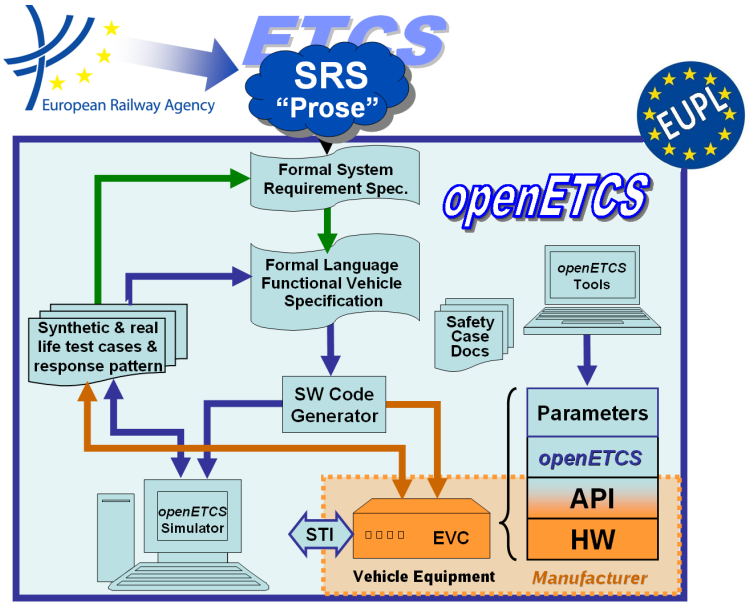
Work-Package 2: "Requirements for Open Proofs"

openETCS D2.1: Report on existing methodologies

State of the art of means of description, methods and tools

Jan Welte and Hansjörg Manz

Januar 2012



Funded by:

Federal Ministry of Education and Research
Région de Bruxelles-Capitale
GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

openETCS

This page is intentionally left blank

Work-Package 2: "Requirements for Open Proofs" **openETCS/WP2/D2.1**
Januar 2012

openETCS D2.1: Report on existing methodologies

State of the art of means of description, methods and tools

Jan Welte and Hansjörg Manz

Technische Universität Braunschweig
Institute for Traffic Safety and Automation Engineering
Langer Kamp 8
38106 Braunschweig, Germany
eMail: openetcs@iva.ing.tu-bs.de

Final Report

Prepared for openETCS@ITEA2 Project

Abstract: This report presents an overview on existing methods used in the railway sector and other comparable industries for system development including verification and validation. The results are based on a number of interviews with experts from different organisations and various fields of expertise needed for system development, as well as the available literature. With account to the Open Proof concept mainly formal and model-based approaches have been examined for this report. Based on the existing methods, a number of requirements for the development method, adequate means of description and used tools have been captured. They have to be satisfied to successful demonstrate the required level of safety for the ERTMS system.

Table of Contents

Figures and Tables

Figures

Tables

1 Introduction

The CENELEC standards requires a systematic approach for the development of systems and software for railway control and safeguarding systems. Therefore the following functional steps have to be conducted based on the overall System and Safety Requirements Specifications in the EN50128.

- Define the Software Requirements Specification and in parallel consider the software architecture (software architecture is where the basic safety strategy is developed for the software and the software safety integrity level);
- Design, develop and test the software according to the Software Quality Assurance Plan, software safety integrity level and the software lifecycle;
- Integrate the software on the target hardware;
- Validate the software;
- Maintain software during the operational life.

Verification assessment and quality assurance have to be applied across all steps of the development process as well. Therefore, a modular and top-down design approach is needed which provides clear and auditable documents for verification and validation. Accordingly, the standard recommends the software development lifecycle and documentation set shown in Figure 1.

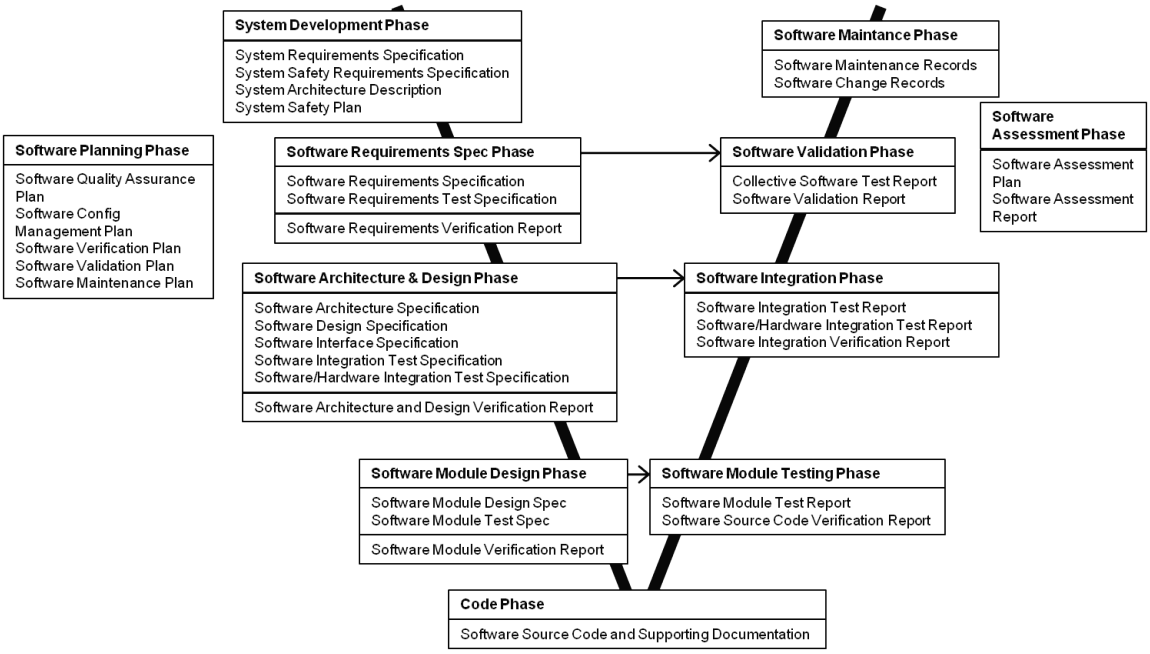


Figure 1. Recommended Development Lifecycle in EN 50128:2011

On the base of this lifecycle the software development tool chain has to be able to provide assistance and generate all needed software and documents for all following eleven phases:

1. System Planning and Development
2. Software Requirements Specification
3. Software Architecture and Design
4. Software Module Design

5. Code
6. Software Module Testing
7. Software Integration
8. Software/Hardware Integration
9. Software Validation
10. Software Assessment
11. Software Maintenance

As the phase feature different tasks and related documents which have to be generated and used by a number of different persons in various roles a structured development process has to be applied. To be able to prove that all safety properties of the ETCS system are captured in the system requirements and software specifications, as well as that the final software product satisfies all specifications, a formal approach is necessary. As the scope of the openETCS project is to allow formal proof for the ETCS on-board software development, formal methods have to be used to provide the basis for formal proofs.

Thereby formal methods is just a generic term for mathematical based procedures covering a number of different functionalities like notations, design techniques and software tools. A more precise classification has to be used to distinguish the needed components and their requirements. We will use the BMW-Concept which separates between the following three means needed to handle tasks [Schnieder, 1999]:

- **B**eschreibungsmittel (ger.): Means of Description (eng.),
- **M**ethoden (ger.): Methods (eng.)
  - Combination of a Means of description and a method represents a technique
- **W**erkzeuge (ger.): Tools(eng.).
  - Combination of a means of description, a method and a tool creates a technology

The means of description is the notation in which all artifacts are clearly modelled and from which a presentation of results can be generated. This can range from a natural language to highly mathematical languages [Schnieder, 2010]. Thereby the degree of formalisation for means of description can be classified by their sigmatic, logical syntax and semantic as it is shown in table 1 [Schnieder, 2010].

Every development process runs through a sequence of tasks with multiple iterations in between and various interactions between the different tasks. The methodology defines the way a tasks is handled by specifying the explicit result of the task and how it is reached. Furthermore to apply the methodology on a task by using the means of description a tool is needed.

In the context of modern computer-based working processes tools are almost exclusively realized as software programs that are used to create system designs and immediately analyse them.

In this way the integration of a suitable means of description (B), the structured process of a methodology (M) and the support of an applicable tool (W) allows to distinguish three orthogonal means, which together provide the needed functionalities, but are independent from each other as far as they can be used separately and exchanged. In practice, a clear distinction between means of description and methodology is not always given, so that both are summarised under the term technique. To follow the structured system development approach of the CENELEC standards in an "open" way means of description, methods and tools have to be chosen, which can be used by everyone.

Table 1. Degree of formalisation for means of description

		Semiotic aspects of means of description			
		Sigmatic	logical syntax	semantic	Example
Degree of formalisation	Informal	Incompletely defined set of symbols	Informal defined	Informal defined	Natural language
	Semi-formal	Completely defined set of symbols	In some cases mathematically defined to be unambiguous, complete and consistent (often defined merely informal)	Informal defined	Message sequence charts and Structured analysis
	Formal	Completely defined set of symbols	Mathematically defined to be unambiguous, complete and consistent	Mathematically defined to be unambiguous, complete and consistent	Programming languages and Petri nets

The CENELEC standards do not demand the use of certain means of description, methods or tools, but give recommendations for all process steps and Safety Integrity Levels (SIL). But the standard requires the use of semi-formal and formal means of descriptions. Functionality and Black-Box tests are mandatory for verification and validation of software with SIL 3 and SIL 4 and highly recommended for lower levels. To validate the final software also performance tests are needed. A traceability of all requirements and specifications over all steps of the development process has to be guaranteed according to the EN50128.

2 Expert interviews

The main source to receive information for this analysis of the existing means of description, methods and tools used for the development of safety relevant railway and comparable systems have been interviews with experts from the railway sector and other industries concerning their approaches and their experiences. Overall 14 interviews with experts from different fields have been conducted at the following companies:

- AEbt (Germany)
- Alstom (France)
- CERTIFER (Paris)
- DB Fernverkehr (Germany)
- DB Netz (Germany)
- ERTMS Solutions (Belgium)
- Formalmind (Germany)
- RATP (France)
- SNCF (France)
- Systerel (France)
- Siemens (France and Germany)
- SRE (Switzerland)

A number of interviews with other valuable experts were enquired but could not be realised due to scheduling problem.

- Airbus (France)
- BMW (Germany)
- Bombardier (Germany)
- SBB (Switzerland)
- BAV (Switzerland)

The interviews have been carried out as a free conversation between the up to three interviewees and the usually two interviewers. Although the main focus for every interview has been chosen according to the company's field of business and the expertises of the interviewees, every interview was guided by the questionnaire presented in Appendix A, which focuses on the following six categories of questions:

A Business and organization of the company

B Methodology, means of description and tools for there software system development process

C Methodology, means of description and tools used for source code generation

D Verification and validation processes for models and code

E Experiences with different methodology, means of description and tools

F Requirements on future methodology, means of description and tools

As most experts are only involved in a small part of the system development process individually not appropriate questions where skipped, but in most interviews all six categories were covered. Subsequent to the interview the provided information were edited and analysed by the interviewers. If necessary, the interviewees have been asked to clarify certain answers or to provide additional information for further research. With this approach the interviews were able to provide a broad overview about methods, means of description and tools which are currently used.

An overview about all relevant means of description discussed during the interviews and their properties is given in Chapter 3. The methods used for the development of safety relevant software systems and eventual constrains concerning their use in the openETCS project are presented in 4. Subsequently chapter 5 introduces the tools used in practice to apply the means of description according to the methods.

Although to not limit the answers to a certain aspect the last questions concerning the requirements for future methodology, means of description and tools where designed in an open way. Nevertheless, most interviewees focused on tool characteristics. This is comprehensible as in practice methodology and means of description are implied by the use of tools. As these tool requirements are indirectly connected to more general requirements for the methodology and the means of description a list of common requirements for all components has been established based on the interviews.

3 Means of Description

In general it can be observed that documents formulated in natural language are ambiguous, since its syntax and semantics always provide room for interpretation for the communication partners. As the means of description provide the basic notations in which the different development task are handled, a wide range of different means of description have been developed to facilitate communication. In general, every mean of description is defined by its symbolism, syntax and semantics which are specified more or less detailed and formal [Schnieder, 2003, 2010].

Depending on the various circumstances under which means of description have been developed and their basic theoretical concepts various characteristics and attributes can be distinguished. The following criteria can be used to characterize means of description based on the guideline 3681:

- Formal basis
- Representation
- Description of structure
- Description of behaviour
- Explicit time representation

Further more the applicability of all means of description in practical applications is influenced by the following three aspects:

- Required expertise
- Level of standardisation
- Tool support

FIXME: *The criteria list should be extendet with deterministic behaviour. Some languages (example: SysML state machines) do not guarantee determinisc behavior of functions due to ambigious semantics. For safety related function determism is an indispensible criterium !*

Corresponding with these criteria means of descriptions are suited to be used in different development phases. Overall the following for levels of abstraction have to be supported:

- System development
- Software requirements and specifications
- Software architecture and design specifications
- Software source code and the compiled object code

Table 2 gives an overview of the relevant properties of means of description and techniques used for system and software development. Therefore also programming languages have been incorporated in this list, which are in general not seen as formal methods. As this list is based on the means of description discussed during the interviews, used in comparable project and the formal methods named in the CENELEC standards, it just provides examples and cannot be seen as complete. The criteria used in table 2 are adopted from the 3681 and based on the important requirements for means of descriptions named during the interviews. The assessment for each means of description is based on the information provided during the interviews, the CENELEC

standards and additional research. For that purpose the instructions manuals and bibliographies of techniques were examined.

Table 2. Characterisation for relevant Means of Description in basis of VDI/VDE 3681:2005

	Criteria									Level of abstraction			
	MoD/Technique	Formal basis	Representation	Description of structure	Description of behaviour	Explicit time representation	No expertise required	Level of standardization	Tool support	System development	Software requirements and specifications	Software architecture and design specifications	Software source code and the compiled object code
ACSL (ANSI/ISO C Specification Language) and C	MoD	o	T	+	+	-	-	+	+		+	+	+
Ada and Spark	MoD	o	T	+	+	-	-	+	+			+	+
Alloy	MoD	+	M	+	+	-	-	-	+	o	+	+	
CNL (Controlled Natural Language)	MoD	+	T, M	-	o	-	o	-	o	o	+		
HOL (High Order Logic)	MoD	+	M	+	+	+	-	-	o	o	+		
Lustre/ Textual and graphical Scade	MoD	+	T, G	+	+	-	-	o	+	o	+	+	+
OBJ	MoD	+	M	+	o	-							
Timed Petri Nets	MoD	+	M, G	+	+	+	-	+	o	+	+		
Process Calculi (CCS, CSP, LOToS)	MoD	+	M	-	o								
RSL (RAISE Specification Language)	MoD	+	M	+	+	+	-	o	+	o	+	+	
State Machines	MoD	+	M, G	-	+	-	-	-	+	+	+		
TL (Temporal Logic)	MoD	+	M	-	+	-	-	-	o			+	
UML 2.0 (Unified Modelling Language) and SysML (System Modelling Language)	MoD	o	T, G	+	+	+	-	+	+	+	+	o	
VDM (Vienna Development Method)	Tech	+	M	+	o	-	-	+	+	o	+	+	
Z, B - Method and Event B	Tech	+	M, G	+	+	o	-	+	+	o	+	+	

MoD - Means of Description, Tech - Technique; T - textual, M - mathematical-symbolic, G - graphical

+ - fulfills criteria completely (can be used) , o - fulfills criteria partially (can be used to some extent), - - does not fulfills criteria (cannot be used)

From the means of description presented in EN50128 only UML, B-Method and Petri Nets were named during the interviews as actually used means of description for modelling the requirement specifications and developing the software architecture and design. In addition State machines and Lustre/ Textual Scade are commonly used for software development. C and ADA and their related languages are primary used as programming languages to write the source code. Therefore these set of means of description can be seen as proven in use for the current software development. The following subsections present further information for every means of description. A more detailed version of the characterisation table is given in Appendix B.

3.1 UML 2.0 (Unified Modelling Language) and SysML (System Modelling Language)

Characteristics

The Unified Modelling Language (UML) an the System Modelling Language (SysML) are object-oriented means of description specified by the Object Modelling Groupe (OMG). Both consist of a number of different diagrams to describe the various aspects of a system. Thereby SysML can be understood an extension of a certain subset of UML developed for system engineering as it uses a part of the UML diagrams. In addition both languages interchanged their models using the XML Metadata Interchange (XMI) standard defined by OMG. Respectively a number of tools are developed to work with both languages.

Typical applications

UML and SysML is dedicated on system engineering and suitable for a number of different kind of systems. UML has be first created to develope software systems, but represents a general approach, which have been adoped in many areas. SysML has be specified to provide a more suited subset of UML for general system development, therefore it is used for the development of all kinds of technical systems.

Standards

UML is defined in ISO/IEC 19501 Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2. It is further developed by OMG and standardised by their publications. OMG also defines the SysML standards.

References

Links:

`http://www.uml.org/`

`http://www.omgsysml.org/`

`http://www.omg.org/`

Bibliography:

- Weilkiens, Tim: Systems Engineering with SysML/UML – Modeling, Analysis, Design, Morgan Kaufmann and The OMG Press, Boston, 2007.
- Ambler, Scott William: The Object Primer – Agile Model Driven Development with UML 2, Cambridge University Press, Cambridge, 2004.

3.2 Petri Nets

Characteristics

Petri Nets were developed originally by Carl Adam Petri in 1962 as graphical way to model chemical reactions. The have evolved since then presenting a very powerful completely mathematical defined means of description to graphically model systems in a discrete way and analyse and simulated their behaviour. In general Petri Nets are bipartite graphs that consist of to kind of notes called places and transitions, which are connected by arcs. Thereby, arcs are only allowed to run from a place to a transition or vice versa and never between places or between transitions. Places represent a local discrete state of the modelled system, while every transition represents a change in these system states. The places from which an arc runs to a transition are input places of this transition and the places to which arcs from the transition run are output places of the transition. That a transition can fire the conditions represented by all input places have to be established. Through the transition firing the system changes from the input conditions to the output conditions by activating the related places. Tokens have been introduced to represent the activated states. Their movement allows to simulated the system behaviour. Over the time various extensions have be developed to extend the concept of Petri ets to handle a larger amount of system properties and behaviour.

Typical applications

Petri nets are used to describe a wide range of systems that show a discrete behaviour, as the overall concept of states and transitions can be found in most application fields. As the Petri Nets concept has been developed and extended over time, the nets have been applied in a number of different fields: Office automation, work-flows, flexible manufacturing, programming languages, protocols and networks, hardware structures, real-time systems, performance evaluation, operations research, embedded systems, defence systems, telecommunications, Internet, e-commerce and trading, railway networks, biological systems.

Standards

ISO/IEC 15909-1 Software and Systems Engineering - High-level Petri Nets Part 1 covers Concepts, Definitions and Graphical Notation Part 2 covers the PNML language, a Transfer Format for Petri nets based on XML

The use of petri nets for the system dependability analysis is standardised in IEC 62551 Analysis techniques for dependability - Petri net modelling.

References

Links:

`http://www.informatik.uni-hamburg.de/TGI/PetriNets/`

`http://www.pnml.org`

`http://www.scholarpedia.org/article/Petri_net`

Bibliography:

- David, René; Alla, H.: Petri nets and Grafcet – Tools for modelling discrete event systems. Prentice Hall, 1992.

- Girault, Claude; Valk, R.: Petri nets for systems engineering – A guide to modelling, verification, and applications. Springer, 2003.
- Van der Aalst, W.M.P. and Stahl, C.: Modeling Business Processes – A Petri Net-Oriented Approach. The MIT Press, 2011.
- K. Jensen and Kristensen, L.M.: Coloured Petri Nets – Modeling and Validation of Concurrent Systems. Springer-Verlag Berlin, 2009.

3.3 Z, B - Method and Event B

Characteristics

All three formal methods have mainly been developed by Jean-Raymond Abrial. The Z notation is the oldest of the three and named after Zermelo–Fraenkel set theory. It has been developed as a specification language which used a model-based approach to describe computing systems. The Z language is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic.

B has been developed as a associated method to Z which is more focused on the lower-level formal code development rather than just formal specification. Respectively the B method provides more than the pure means of description, as it additionally includes processes for refinement, proof and has a strong tool support. Normally the B method development of software first writes a concrete model which specifies the main data processes of the system and the fundamental properties of this data. Thereby the languages used set theory using invariants to describe the static properties and stating post-conditions for operations to define the dynamic behaviour. The B model is then refined to obtain a complete software model, which can be formally analysed to prove the correctness.

The newest formal method in this group is Event-B which has been developed as an evolution of B. Event-B has a simplified notation aiming at system-level modelling and analysis.

Typical applications

Z and B are used in a variety of applications to develop safety critical software since the models allow to prove the correctness. The main use is in the railway and aerospace industry. As Event-B is respectively new it has not yet be applied on major projects.

Standards The Z method is defined by the standard SO/IEC 13568: Information technology – Z formal specification notation – Syntax, type system and semantics.

The B method and B Event are mainly based on the books published by Jean-Raymond Abrial. These are the quasi standardisation of the means of description.

References

Links:

`http://www.methode-b.com/`

`http://www.event-b.org/`

Bibliography:

- Spivey, J. Michael: The Z Notation – a reference manual, Prentice Hall International Series in Computer Science, 1992.
- Abrial, Jean-Raymond: The B-Book – Assigning Programs to Meanings, Cambridge University Press, 1996.
- Schneider, S.: The B-Method – An Introduction, Palgrave Macmillan, Cornerstones of Computing series, 2001.
- Abrial, Jean-Raymond: Modeling in Event-B – System and Software Engineering, Cambridge University Press, 2010.

3.4 Lustre/ Textual and Graphical Scade

Characteristics

SCADE is a formal modelling language targeted for safety-critical embedded control applications in the avionics, rail, automotive and industrial automation domain. SCADE source code can be written as text (for anyone who likes writing plain text) or (more usual) as schematic diagrams. SCADE models are synchronously clocked data flow and state machines, that can be nested and intermixed with each other without limitations. SCADE provides DO-178B- and EN50128-certified code generators producing C or ADA code as output. SCADE models are therefore concrete, deterministic, executable and verifiable; it allows the production of rapid prototype as well as of safety related target system software.

Typical applications

Safety critical systems like

- Rail interlocking systems
- Rail track vacancy detection systems
- Rail train control systems
- Rail Level-crossing protection systems
- Avionic flight controller

Standards

The languages Lustre and SCADE are well documented, but note standardised.

References

`http://www.mobility.siemens.com/mobility/global/en/interurban-mobility/rail-solutions/rail-automation/Pages/rail-automation.aspx/`

Links:

`http://esterel-technologies.com/`

`http://www.interested-ip.eu/`

`http://http://www.interested-ip.eu/final-report.html/`

Bibliography:

- Caspi, P. ; Pilaud, D.; Halbwachs, N.; Plaice, J. A.: LUSTRE – A declarative language for programming synchronous systems,1986.

3.5 State Machines

Characteristics

State machines and the for software development most relevant subset of finite-state machine represent the major mathematical concept to describe a system structure and the behaviour. Thereby states are defined which represent a certain status of the system, while inputs are information triggering certain actions leading to a state change. If the system actions are defined for every input in every state, the system is described completely.

If the number of states and transitions is fined and countable, a start state defined and a set of final states given, the system is described by a finite-state machine. The limits of a finite-state machine all to graphically represent the system in a state transition diagram showing the overall behaviour and to check the finite-state machine for completeness, consistency and reachability.

Typical applications

Most means of descriptions developed to describe a system behaviour incorporate the principle of state machines in their overall concept.

Standards

As state machines are a pure mathematical concept, their is no standard. They are defined in basic automata theory.

References

Links:

`http://foldoc.org/state+machine`

Bibliography:

- Carroll, J., Long, D.: Theory of Finite Automata with an Introduction to Formal Languages. Prentice Hall, Englewood Cliffs, 1989.
- Ginsburg, S.: An Introduction to Mathematical Machine Theory. Addison-Wesley, 1962...

3.6 VDM (Vienna Development Method)

Characteristics

The Vienna Development Method has been developed in IBM's Vienna Laboratory in the 1970s as a formal specification language. It is a model-based approach which describes the system by its states using set-theoretic structures. Thereby, the pre and post conditions of a state are defined as invariants and operations. These can be proven, while the data structure of the system is set up during reification and the operations ae refined.

Typical applications

VDM has been used in industry to develop proven software for critical systems, compilers and concurrent systems

Standards

The VDM has be standardised in ISO/IEC 13817 Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language.

References

Links:

`http://www.vdmportal.org`

`http://www.vdmbook.com`

Bibliography:

- Bjørner, D. and Jones, C.B..: The Vienna Development Method – The Meta-Language, Lecture Notes in Computer Science 61, Springer, 1978.
- Bjørner, D. and Jones, C.B.: Formal Specification and Software Development, Prentice Hall International, 1982.
- Jones, C.B.: Software Development – A Rigorous Approach, Prentice Hall International, 1980.

3.7 Process Calculi (CCS, CSP, LOToS)

Characteristics

Process calculi (or also named process algebras) are a group of approaches to formally model concurrent systems to specify the processes and verify their implementation. The model the system by specifying the high level independent processes and their communication. Thereby, the process can be sequential or parallel. The modelling is done by using a pseudo-programming language. This allows to use algebraic lows for manipulation, analysing and formal reasoning.

Typical applications

These means are mainly used to mathematically model and analyse communications in software systems.

Standards

The different process calculi are documented in the respective manuals, but their is no overall standardisation.

References

Links:

`www.usingcsp.com`

`http://theory.stanford.edu/~rvg/process.html`

Bibliography:

- Hennessy, M.: Algebraic Theory of Processes, The MIT Press, 1988.
- Hoare, C. A. R.: Communicating Sequential Processes, Prentice Hall, 2004.
- Milner, R.: A Calculus of Communicating Systems, Springer, 1980.

3.8 HOL (High Order Logic)

Characteristics

Higher-order logic is a predicate logic notation which distinguished from first-order logic like set theory by additional quantifiers and a stronger semantics. This allows a instead, more immediate and natural representation of system properties, which has to be been done be encoding in a first order logic.

Typical applications

Higher-order logic is applied on software applications for tasks like natural language parsing and theorem proving. In addition computational higher-order logics have alsobeen used as meta input languages for the application of theorem provers.

Standards

Since High Order Logics are a purely mathematical concept, they a well documented and standardised with respect to the mathematical theory.

References

Links:

`http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/AIencyclopedia/`

`http://plato.stanford.edu/entries/logic-higher-order/`

Bibliography:

- Church, A.: A Formulation of the Simple Theory of Types, Journal of Symbolic Logic 5 56 – 68, 1940.
- Shapiro, S.: Foundations without Foundationalism – A Case for Second-Order Logic, Oxford University Press, 1991.
- Andrews, P. B.: An Introduction to Mathematical Logic and Type Theory – To Truth Through Proof, Kluwer Academic Publishers, 2002

3.9 TL (Temporal Logic)

Characteristics

Temporal logic extends the first order logic concept in this way, that modal relations can be expressed. This concept of temporal logics is based on the work of Arthur Prior. This can only express modal aspect between state like some state changes "always" or "eventually" not quantified time relations and constraints. Absolute time can be introduced by using specific time states for the system modelling.

Typical applications

Temporal logic is used for the formal verification of hardware and software systems asthe state requirements can be specified in temporal logic. Thereby, temporal logic allows direct specification of safety and operational properties concerning the system behaviour, which can then be formally demonstrated for the following development steps.

Standards

Since Temporal Logic is a purely mathematical concept, it is a well documented and standardised with respect to the mathematical theory.

References

Links:

`http://plato.stanford.edu/entries/logic-temporal/`

Bibliography:

- Kamp, J. A. W.: Tense Logic and the Theory of Linear Order, Ph.D. thesis, University of California, Los Angeles, 1968.
- Prior, A. N.: Time and Modality, Clarendon Press, Oxford, 1957.
- Øhrstrøm, P. and Hasle, P.: Temporal Logic – From Ancient Ideas to Artificial Intelligence, Dordrecht, Boston and London Kluwer Academic Publishers, 1995.

3.10 OBJ

Characteristics

OBJ is a group of algebraic programming and specification languages, which use requirements in terms of algebraic equations enriched with other logics to specify the system. The specification can be formally proven. OBJ languages can only be applied to sequential systems.

Typical applications

OBJ languages can be used to precisely specify sequential systems and through the precise algebraic specification execute the specification to receive analysis and system validation prior to the actual implementation.

Standards

The OBJ languages are defined by their manuals.

References

Links:

http://cseweb.ucsd.edu/~goguen/sys/obj.html

Bibliography:

- Goguen, J. A. : Higher-Order Functions Considered Unnecessary for Higher-Order Programming. In Research Topics in Functional Programming, 1976.

3.11 CNL (Controlled Natural Language)

Characteristics

As the name implements the controlled natural languages (CNL) represent a subsets of natural languages. These restrict the grammar and vocabulary of a natural language to to reduce and/or eliminate ambiguity and complexity existing in a natural language. In general two main categories of CNL exist. The one category has the aim to improve the readabilty through restriction to enable easier understanding or allow automatic analysis of the semantic. The second category limits the language to reach a formal logical basis, which can be conected to an existing formal means of description.

Typical applications The first category is often used to reach more consistent documents, which generates higher quality documents and automatic translation. The second category provide formal texts which can be handled automatically by software tools.

Standards

No specific standard.

References

Links:

https://sites.google.com/site/controllednaturallanguage/

Bibliography:

- O'Brien, S.: Controlling Controlled English – An Analysis of Several Controlled Language Rule Sets, Proceedings of EAMT-CLAW, 2003.
- Tonetta, S.: Extending the Property Specification Language with a First Order Signature, Fondazione Bruno Kessler IRST, tech rep. number fbk161120071, 2007.

3.12 Alloy

Characteristics

Alloy is a declarative specification language based on the notion of relations for expressing complex structural constraints and behaviour in a software system. Alloy models are micro-models which can be checked automatically for correctness. Alloy models consist of different

kinds of statements. Signatures are the basic sets like objects or lists. A fact is a constraint that has to be guaranteed. Predicates are parameterized constraints, while functions are expressions giving back a result. Assertions are logical expression which are used to check whether the model holds these properties.

Typical applications

The alloy specification language is mainly used for tool development.

Standards

The book by Daniel Jackson is the main standard for alloy. **References**

Links:

`http://alloy.mit.edu/alloy/`

`http://www.doc.ic.ac.uk/project/examples/2007/271j/suprema_on_alloy/Web/`

Bibliography:

- Jackson, D.: Software Abstractions – Logic, Language, and Analysis, MIT Press, 2006.

3.13 Ada and Spark

Characteristics

Ada is programming language which uses a structured, statically typed, imperative and object oriented approach. The language was originally aimed embedded and real-time systems, but has been extended over time. Ada is specifically designed for the development of large software programs and has already a number of checks incorporated to detect bugs.

SPARK is a evolution of ADA, which has been develop to eliminate the ambiguities and insecurities in ADA to provide a formal programming language for safety-relevant software.

Typical applications

ADA and specifically SPARK is used to write safety critical software in various areas like nuclear, aerospace or railway applications.

Standards

The Ada version 2012 is defined by the ISO/IEC 8652 Information technology – Programming languages – Ada.

References

Links:

`http://libre.adacore.com`

`http://www.adaic.org/`

`http://www.ada-auth.org`

Bibliography:

- Barnes, John: High Integrity Ada: The SPARK Approach, Addison-Wesley, 1997.

3.14 ACSL (ANSI/ISO C Specification Language) and C

Characteristics

The C language is a imperative programming language, using structured programming and a static type system. As the C language structure can be efficiently translated to machine instructions, it is commonly used for software like operation systems. Overall the C language is the most used programming language, therefore it has been used with most hardware architectures and operating systems.

The ANSI/ISO C Specification Language (ACSL) is a language developed to write specifications using pre- and postconditions and invariants to develop C programs respecting these contracts. The specifications are written as annotation comments inside the C code, which is then compiled.

Typical applications

The C language is used for all kind of programs and is specifically efficient for software close to the hardware architecture. Respectively the C language is used for C is often used for implementing operating systems and embedded system applications. In some cases C is used as an intermediate language for other language implementations for convenience or portability.

ACSL is used to specify certain behavioral properties of a C code, which can then be verified a tool through deductive reasoning.

Standards

The C language has been standardised in the ISO/IEC 9899 Information technology – Programming languages – C. Depending on the publication year the current version is called C11.

References

Links:

`http://frama-c.com/acsl.html`

`http://www.open-std.org/jtc1/sc22/wg14/`

`http://cm.bell-labs.com/cm/cs/who/dmr/chist.html`

Bibliography:

- Banahan, M.; Brady, D.; Doran, M.: The C Book, Addison-Wesley, 1991.
- Ritchie, Dennis M.: The Development of the C Language, The second ACM SIGPLAN History of Programming Languages Conference, p.201–208, 1993.

3.15 RSL (RAISE Specification Language)

Characteristics

The RSL is the specification language, which is developed as main component of the RAISE (Rigorous Approach to Industrial Software Engineering) method developed in the 1990s by the ESPRIT II LaCoS project, led by Dines Bjørner. As a specification language the RSL provides a rich, mathematically based notation to formulated and reasoned about requirements, specifications and to some extend steps of the software design. To allow such a wide-spectrum of formulations the RSL uses abstract, axiomatic styles of description as well as concrete, operational styles. Overall the language can be used from initial domain and requirements analysis through design to a level of abstraction from which source code is generated.

Typical applications

RAISE has been used for different kind of engineering software project e.g. to handle civil engineering project or to model a railway interlocking.

Standards

The RSL and the RAISE method are mainly standardised through the books published by the The RAISE Language/ method Group.

References

Links:

`http://www2.imm.dtu.dk/~dibj/raise/`

Bibliography:

- The RAISE Language Group: The RAISE Specification Language, BCS Practitioner Series, Prentice Hall, 1992.
- The RAISE Method Group: The RAISE Development Method, BCS Practitioner Series, Prentice Hall, 1995.

3.16 Summary

This selection of various means of description used in the railway industry shows that the notations have been developed with different targets so there is probably no single means of description able to support all levels of abstraction and to describe all structural and behavioural aspects of a system. Therefore a combination of means of description is needed to support all development steps and to capture all aspects of an ETCS on-board system. The advantages of using different means of description for an easier understanding or a more exact notation of certain aspects have to be evaluated against the difficulties of coordinating the different means of description. To ensure consistency over all levels of abstractions when means of description are changed the area of use for every means of description has to be clearly defined and validated automatic translation should be used. Overall the chosen means of description has to provide the needed formal notation to apply the methodology.

4 Methods

Usually one methodology is used to handle one step in the development process. If a methodology can cover only a certain aspect of a development step certain methods can be combined. The Appendix A of EN50128 lists a number of methods for every step of the software development life cycle and gives recommendations which methods to use. Especially for the verification an validation steps combinations of methods are necessary. According to the standards the software requirements specification have to be provided in natural language and if necessary in a formal/ semi-formal description.

As the openETCS project looks for methods to incorporate the open proof approach into the software development for the ETCS on-board unit, the characteristic of proofability has to be considered for all determined software requirements. Since the system requirements specifications are provided in a textual way, these have to be translated into a formal description to allow for mathematical reasoning and to establish proofable software specifications. During the development this formal description has to be translated back into natural language to provide all kinds of needed documentation. Overall suitable methods for the following seven general functionalities of the software development process have to be found:

- Transformation of textual specifications in formal specifications
- Transformation of formal requirement specifications in formal software and software module design and architecture descriptions
- Source code generation
- Verification of models and source code
- Validation
- Creation of documentation
- Terminology management/Intelligent Glossary

The interviews have shown that these tasks are normally divided in a number of subtasks according to the used methodology. Respectively a broader methodology defines the process to deal with the task and more specific methods describe the actual work steps. This approach is backed through the recommendations given in the EN50128. The following sections present the methods applied in practice to handle the six general tasks.

4.1 Transformation of textual specifications in formal specifications

As a formal specification is build using a formal mean of description, their properties mainly defines the actual process to translate the natural language requirements into a formal notation. Thus, this task is usually handled by experts who are familiar with the formal means of description and try to formulate the formal description as the read the textual specification. As this process depends on the individual knowledge and understanding of the expert the interviews have not been able to cover those work steps in all details. But in generically the two following subtasks had to be completed for every textual requirement to provide a complete formal model:

- Informal analysis: Collect and categories the requirements stated in text
- Formalization: Allocated the category of requirement to a concrete formal representation and put these in relation to all other already formalised requirements

In some case a generic list of specific categories of requirements and their allocated concept of formal representation have been established at the beginning, but mostly these steps are done more or less consciously during the translation.

The methodology described in [Cimatti et al., 2008] can be seen as representative way how the textual system requirement specifications covered in subset 26 can be translated systematically into a formal/semi-formal description. In this case the chosen methodology is based on UML diagrams combined with CNL, which allows subsequently to verify the formal specification through model checking.

During the informal analysis step the textual specifications are decomposed in basic requirements which are collected using an appropriate tool to store them. These basic requirements can then be classified by assigning one of the eight categories from table 3.

Table 3. Categories of requirements according to [Cimatti et al., 2008]

Category	Description
Glossary	Defines a particular term/concept
Architecture	Introduces some system's modules and their interactions
Functionality/behaviour	Describes the behaviour a module can perform or the states a module can be in
Communication	Describes massages exchanged between modules
Environmental	Describes constrains on the model
Scenario	Describes possible scenario
Property	Describes expected/required properties
Annotation	Notes in the specification

Afterwards the basic requirement is expressed by using the associated UML or CNL formalisation which is linked back to the textual requirement to provide traceability. The methodology in [Cimatti et al., 2008] for example uses UML state machines to formalise functional/behavioural requirements and CNL to specify environmental requirements.

Independent of the concrete methodology and the used means of description the interviews have shown that it is central to guarantee the traceability of all requirements as the completeness of the software can only be validated if it is documented that the entire textual specification is covered. Therefore the traceability has to be established from the source code to the formal specifications and from there back to the textual specifications. Differences between the textual and the formal specification have to be reasoned by linking it to inconsistencies or incompleteness in the textual specifications.

4.2 Transformation of formal requirement specifications in formal software and software module design and architecture descriptions

To write their formal software requirement specifications and to define the software architecture and design based on these most organisations presented a model-based development process using different formal models to describe the software on different abstraction levels. In doing so the railway operators mostly focus on the high-level description of system and software functionality, while the actual software developers concentrate on modelling the more concrete software architecture and design. By using formal methods and formal refinement certain steps of the top-down modelling can be automated. Since every organisation used a modular approach as it is mandated by **??** for all SIL Levels the refinement is also used to separate the functionalities of

different modules. Thereby the detailed software model can than be used for an automatic or semi-automatic source code generation. To model the interactions with adjacent systems including human operators a model representing the environment is needed. Overall our interviews have confirmed that a variation the top-down model-based software development shown in figure 2 is common practice at least for the development of safety related software.

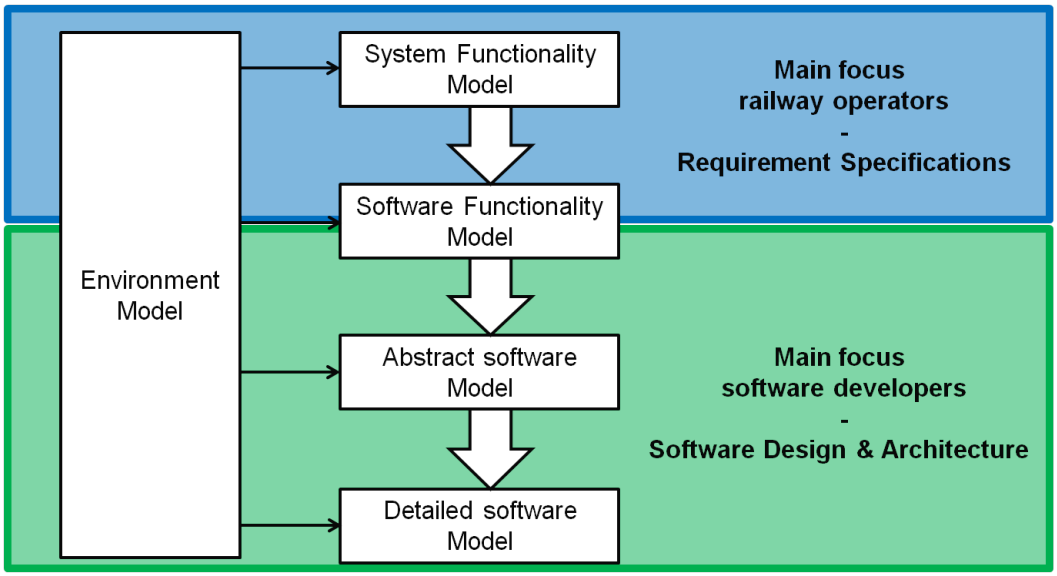


Figure 2. Model-based development approach for software development

To guarantee certain modelling standards all organisations had defined training processes for their experts which introduced them to certain modelling standards building the mutual basis for all models. Additionally most organisations use models that can be analysed and/or simulated during the modelling process to receive feedback from these methods about certain properties of the model during their model-based development process.

It has already been shown in chapter 3 usually different means of descriptions are used to build functionality models than design and architecture models. As most organisations are focus either on high-level functional models or on design and architecture models they use only one means of description for all models in their internal model-based development process. But usually functionality models which are handed done in the development process for example from an operator or train manufacturer to the software developer have to be translated. Therefore a dependable translation methodology is needed to ensure that all functionalities are incorporated in the software models and to guarantee traceability over all levels of abstraction. In practical applications this is mostly handled by validated translation tools and though corresponding test cases.

4.3 Source code generation

Using the model-based development approach most interviewed organisations generated source code based on their software design and architecture models. As the methodology for source code generation depends explicitly on used means of description and programming language the interviews have not been able to cover them in detail. Especially as most commercial modelling tools for UML, B-Method, SCADE and Simulink already provide an incorporated methodology for automatic code generation.

In most cases ADA and C and their related programming languages are used for source code generation. The source code is then compiled into executable software, in doing so the compiler

has to be validated to guarantee that the software behaviour is according to the source code definition.

Another methodology to implement the functionality expressed in a model into a software is to use an interpretor .

The interpretor is a generic software build to understand the functionality of a certain kind of model which then hands the commands created in the executable model directly down to the hardware controls. Therefore the functionality can be change by just exchanging the model with any changes at the interpretor. To guarantee that the interpretor can provide the wanted functionality he has to understand every possible model behaviour and to convert this into all required commands. Like every used translator or compiler the interpretation has to be validated to ensure a dependable execution.

This methodology which absolutely separates functionality and lower level control parts is currently used for interlocking-systems to guarantee a manufacturer independent programming of functional rules. In this process a specific kind of Petri net models is used to describe the needed system behaviour for every interlocking. If a new behaviour is required only the Petri nets model has to be modified.

As such processes of formal automatic source code generation or model interpretation requires a large effort and completely validated tools this methodology is in the majority of cases only used for safety relevant software. For non-safety relevant software semi-formal code generation is mostly applied to reduce the costs even if this usually creates more bugs.

4.4 Verification of models and source code

To verify the different models and the source code basically to groups of methods can be used:

- Formal analysis and proof
- Testing

While testing reasons that the required defined properties are present when the model or the software shows the wanted behaviour in a number of scenarios, formal analysis and proof use the formal basis of the means of description to reason that the stated properties cannot be violated by the model or source code. Accordingly the formal methods can only be applied if model or source code are formal and the properties, which have to be verified are stated in a formal description.

The properties a system has can be distinguish into the following to categories:

- State/Transition properties,
- Functional properties,
- Structural properties,
- Behavioural properties. /citepSchnieder.2010b

Suitable methods have to be chosen to verify all four kinds of properties.

Generally formal proof, testing and simulation are general principals covering a number of more concrete methods, which have to be chosen concerning the specific properties that have to be verified through the principal.

4.4.1 Formal analysis and proof

Since formal proofs are able to demonstrate the general existence of a property based on mathematical reasoning, a formal proof is able to replace testing if it can be applied. Correspondingly, the interviews have shown that it is already common practice for train control systems to verify the software design and architecture almost exclusively through formal proof as it is illustrated in figure 3.

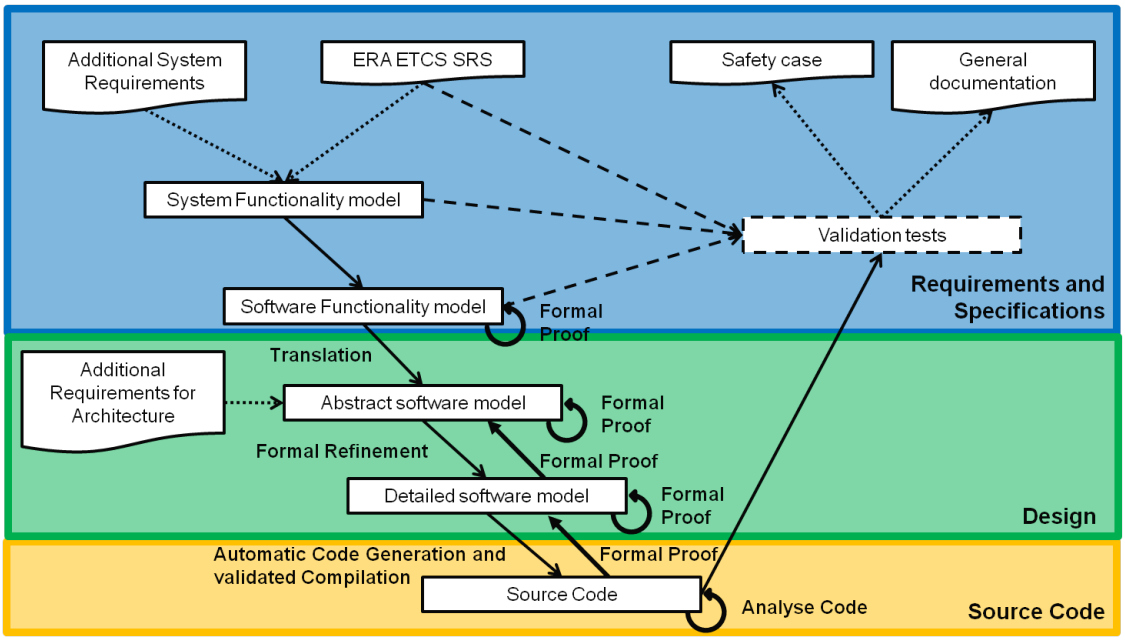


Figure 3. Model-based development with verification through Formal Proof and additional Testing for validation

To apply formal analysis and proof basically three different methods can be used:

- Theorem Proving,
- Model Checking, and
- Simulation.

As a theorems is a mathematical statement which has to be proven based on the basis of certain assumptions theorem proving as a methodology able to verify properties also for infinite systems. Respectively, the ability to prove a theorem is based on the mathematical system description which implies that there can be cases in which a theorem can neither be proven nor disproved. Therefore in some cases it is necessary to adopted the formal description to make certain properties provable.

Model checking uses the formal structure of a model to prove that certain states or variable values which are connected to certain properties can or can not be reached. Therefore the model-based description is limited to a finite number of system states. If a property is rejected a counterexample is provided.

If the mathematical description of all specifications can be simulated it is possible to analyses the system properties according to possible inputs. This can be used to proof, that under all inputs the system shows the required properties. Usually this is used to show certain behavioural properties.

In general formal analysis and proof can only be applied for properties which can be stated in a formal way that is consistent with the mathematical theory used for the underlying model description. This can be challenging if abstract properties shall be proven for a detailed software design. Therefore the interviewees using formal prove for their software verification emphasised

especially that the main part of the work is to define the properties to prove and to refine them for every abstraction level.

4.4.2 Testing

Test methods range from simple checklists to highly complicated Test scenarios. To verify by test that a model fulfills certain requirements a tests cases have to generated which describe what has to be done during the tests and which result is expected. As for complex systems test cases can never cover all possible scenarios testing cannot provide a full guarantee that a properties holds under all circumstances.

Tests can be a static analysis which just ensures that certain aspects are covered by the model or source code or it can be a dynamic test which sets some starting conditions and expects that the model or code responds in a certain way. The simplest kind of dynamic tests are Funtional/Black-Box Tests which just specify inputs and expected outputs without any information about the actual system structure. These are mandatory for the verification of SIL 3 and SIL 4 software, but can be enhanced trough a more precise test case specification as more are available.

These test cases can be modelled by an expert based on the modelled or textual requirements that should be verified or they could be generated in a automatic way based on a formal requirement model. As most organisations we have interviewed use a model-based development for their software development those how used testing for verification applied model-based testing by simulation of their models and creation of test cases based on these models. Figure 4 shows the process structure for a model-based development process with testing for verification and validation.

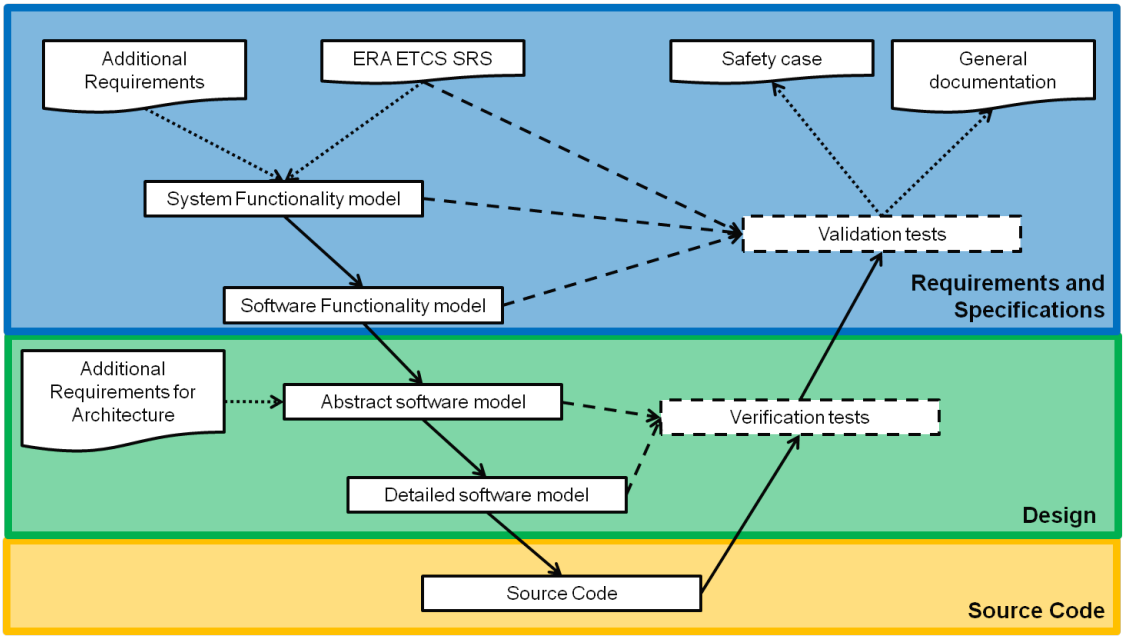


Figure 4. Model-based development with verification and validation through testing

In general test cases are collected in catalogue against which every new version of the model or code is tested. To reduce the amount of work this is usually automated by using a test environment which combines the test case catalogue with an execution tool and provides a test report at the end.

4.5 Validation

Asked for the validation methodology the interviewed organisations explained that the following three methods are mandatory for validation according to EN50128:

- Traceability for functional and non-functional Requirements
- Functional/Black-Box tests
- Software performance tests

As verification and validation are closely related in product development most organisations which used formal proof for verification also this methodology for tasks in their validation process. But as validation has to refer back to the original textual system requirements as every organisation we interviewed argued it can not be done entirely by formal proof. Accordingly a combination of tests and formal analysis and proof is used by these organisation to validate their software as it shown in figure 3.

Overall the model-based development process itself already provides an important methodology validate the software against it requirements. The core of every validation process presented during the interviews has been modelling combined with traceability of all requirements from the original textual system specifications and a suitable catalogues of test cases created based on this textual specifications and their formal models.

4.6 Creation of documentation

Since the software development requires supporting documentation for operation, maintenance as well as for all safety related activities it is important to use a methodology the way all these documents are created. This involves not only a structure when and how documents are created and used which is mainly specified by CENELEC standards but also a methodology how the formal/semi-formal software descriptions are included into the documentation. As natural language documents a methodology is needed which that formal/semi-formal documents and textual documents are consistent.

Most organisations we interviewed already use tools which can automatically create documents based on their formal/semi-formal models. Usually these documents list the processes modelled including their input and outputs and some important modelling characteristics. If possible graphical representations of those models are added to illustrated the modelling work and provide additional information.

As the kind of information, which have to be extracted from specific models for a proper documentation can at least to a certain degree be universally determined it is common practice to create some parts of the textual documents automatically. Especially UML diagrams allow to incorporate specific variables from a diagram into a text by using predefined textual building blocks. This guarantees that the documentation is consistent with the model-based development activities and helps to reduce the needed amount of work to create the documentation as human editing is only needed to check the comprehensibility and to add parts in natural language that cannot be created from the models.

4.7 Terminology management/Intelligent Glossary

The used terminology has a central role in a development process to ensure that all participants understand the requirements in the same way and that these requirements are treated consistently during the whole development process [Schnieder, 2010]. To provide an overview about the specific terminology used to describe a system it is common practice to establish a glossary and

a collection of abbreviations which provide definitions for most of the specific terms. Usually this glossary is just an additional documents for the textual system requirement specifications, which just provides a short explanation for every key term regarding to the concept that term is used in the requirement specifications.

Some interviews emphasized to ensure a consistent terminology over all models and all process participants the relations between different terms have to be modelled, too, and these model has to be link to all models and documents to provide the right terminology to every process step, to allow additions and to provide a traceability over all abstraction levels. Some tools, especially those using UML or SysML, provide some a way to model the terminology and to connect it to other kind of models but no organisation we interviewed used a methodology providing all features of an intelligent glossary for the whole development process. The *iglos* methodology which has been developed in an academic context can be used to clearly separate the meaning of terms in different domains and to allocated them so certain applications. Different kinds of relations can be modelled and the terms can be linked to sources and applications in the textual documents or the modelled system descriptions.

Overall the model-based development process requires an terminology managements methodology which provides the functionality of an advanced glossary gathering, handling and providing a systematic structure for the terminology to all parts of the development process. Therefore the used methodology has to be aligned with all means of descriptions, methods and tools used during the development process.

4.8 Summary

Overall the interviews and the additional research have shown, that model-based approaches are common for all kind of safety relevant software development processes. Thereby some organisations already use such formal models to predominately verify their software design and architecture through formal proof. However, for validation a combination of formal proof and different testing methods is needed as formal proof can only been used for properties which can be state in a suitable way. A tabular overview about all presented methods is shown in appendix **??**. The following chapter 5 presents the tools used in practice to apply these methodology by making use of the different means of descriptions named in chapter 3.

5 Tools

As it has already been described the tools are closely connected to a means of description and a methodology which they support. Therefore a large number of different tools from various backgrounds are available which mostly handle on means of description and allow to apply one or more methods for different steps in the development process.

5.1 Tool use

The interviews have shown that there are tools available and in use to support the following development steps and the connected methods depending on the general development process:

- Formalisation of textual requirements
- Modelling
  - System/software structure
  - System/software behaviour
  - Software architecture and design
- Formal refinement
- Model translation
- (Automatic) Code generation and compilation
- Formal Proof
  - Theorem proving
  - Model checking (different levels)
  - Simulation
- Analysis of source code
- Testing
  - Test environment (for model and software tests)
  - Test case generation and test case database
- Traceability of requirements
- Versioning and configuration management
- Terminology Management
- Documentation

Since the functionalities tools support can be used in different methods and for different development steps the use of a tool can be necessary at different times during the product lifecycle. Additionally some tools are able to provide a range of functionalities so they can be used to combine different methods. Accordingly it is often difficult in practical applications to clear distinguish between tool functionality and supported methodology. Therefore most of the interviewees have focused on the main tools there are using for their model-based development and directly linked the tool to a step of their development process. An overview about the tools discussed during the interviews or presented in related publications is given in table 4. The table also shows the development task which are supported by these tools . This table only covers the

main tools named as relevant for the open ETCS project. Most tool developers provide a number of additional tools which interact with the main tool to provide a range of functionalities. These tools are not listed separately to keep the table focused on main tools.

Table 4. Overview of Tools and the support development processes

Tool	Developer	License status	Means of Description	Formalisation of textual requirements	Modelling	Formal Refinement	Model translation	Code generation and compilation	Formal Proof	Analysis of source code	Testing	Traceability of requirements	Versioning and configuration management	Intelligent glossary	Documentation
Alloy 4	Software Design Group at MIT	free	Alloy		+	+		+	+				+		o
Artisan Studio	Atego	commercial	SysML und UML 2.0	o	+		+	+				+	+	o	+
Atelier B	ClearSy System Engineering	free	B, Event B		+	+	+	+	+				+		o
CompoSys	ClearSy System Engineering	commercial	Event B	o	+				+		+		+	o	+
Control Build	Geensoft	commercial			+			+			+		+		o
CPN	Eindhoven University of Technology	open source	Petri Nets		+				+		+		+		o
Enterprise Architect	Sparx System	commercial	SysML und UML 2.0		+			+			+	+	+		+
ERTMS Formal-Specs	ERTMS Solutions	open source	DSL		+		+	+			+	+	+		o
Frama-C	CEA-LIST and INRIA-Saclay	open source	C						+	+					o

+ - can be used , o - can be used to some extent

Overview of Tools and the support development processes - Continued on next page

Tool	Developer	License status	Means of Description	Formalisation of textual requirements	Modelling	Formal Refinement	Model translation	Code generation and compilation	Formal Proof	Analysis of source code	Testing	Traceability of requirements	Versioning and configuration management	Intelligent glossary	Documentation
iglos	TU Braunschweig, iVA	free/ commercial	Mult. MoD	+								+	+	+	+
iLock/iCertifier	Prover Technology AB	commercial			+		+	+	+	+	+		+		+
KNOW Enterprise	KnowGravity Inc.	commercial	UML 2.0 (xUML)	o	+		+	+			+	+	+	o	+
MagicDraw	No Magic	commercial	SysML und UML 2.0	o	+			+			+	+	+		o
mCRL2	Technische Universiteit Eindhoven	free	mCRL2		+	+			+						o
Modelio	Modeliosoft	open source	SysML und UML 2.0	o	+			+			o	+	+		o
NuSMV	Fondazione Bruno Kessler	open source	BDD and SAT						+						o
Papyrus	CEA	open source	SysML und UML 2.0	+	+						+	+	+		o
Perfect Developer	Escher Technologies Ltd.	commercial			+			+	+		+		+		o

+ - can be used , o - can be used to some extent

Overview of Tools and the support development processes - Continued on next page

Tool	Developer	License status	Means of Description	Formalisation of textual requirements	Modelling	Formal Refinement	Model translation	Code generation and compilation	Formal Proof	Analysis of source code	Testing	Traceability of requirements	Versioning and configuration management	Intelligent glossary	Documentation
PRISM	University of Oxford	open source	BDD and MTBDD						+						o
ProB	Formal Mind GmbH	open source	B						+		+		+		o
ProR	Formal Mind GmbH	open source	ReqIF 1.0.1	+									+		+
Rational Architect	IBM	commercial	SysML und UML 2.0	+	+		+	+			+	+	+		+
SCADE Suite	Esterel Technologies S.A.	commercial	Lustre / Data Flow (Logic) + State machines		+	+	+	+	+	+	+	+	+		+
Simulink/ Design Verifier	MathWorks	commercial	Block diagramms		+	+	+	+	o		+		+		o
SPARK GPL/ GNATprove	AdaCore	open source	Spark ADA		+			+	+	+	+		+		o
SPIN	Bell Labs	open source	PROMELA						+	+	+	+			o
π-Tool	IQST	commercial	Petri Nets		+	+	+		+		+		+		o

+ - can be used , o - can be used to some extent

The focus of the tool discussion during the interviews has been on modelling tools as the collection of tools in table 4 shows. Most companies use commercial tools like Artisan Studio, Atelier B, Rational Architect, SCADE or Simulink to build their software models by using the implemented means of description and the implemented methods to generate reliable source code in C or ADA languages.

To use formal proof for these models usually on of the available commercial tools which fits to the modelling tool is chosen and than customised to a degree that it provides all needed functionalities. Usually this is handled in cooperation with the tool developer. Some companies have a lot of experience with tools for modelling and model proofing and therefore mainly handle the customisation in house.

But especially for proofing and source code analysis a large number of open source tools exist which usually have an academic background. Therefore most of them are not certified. But as some of these tools have been around for quite some time, they could be suitable to fulfil all requirements according to the CENELEC standards. As most interviewees confirmed open source software combined with support through a external partner could provide the needed level of customisation and service support required from most companies. Nevertheless most interviewees raised issues that it could be difficult to get broadly accepted and qualified tools for formal proof and code generation as open source products. Since there is a number of open source tools available for all task in the development process an in deeps evaluation is needed.

5.2 Tool chains

As this overview only covers the most relevant tools covered during the interviews a longer list of all tools mentioned during the interviews is presented in appendix **??**. Especially for formal proofs there is a number of tools which have been developed in academically research to check one or a group of specific properties of a certain kind of models. Most of these tools are open source. If a specific methodology has been choose it is necessary to analyse which of these tools fits the needed verification and validation activities.

Since certain methods like model-based testing include an number of different activities which are supported through different tools it is important for the development process to allow interactions between the tools. This can be the case by allowing data exchange between different tools of specified interfaces or by incorporation certain tools functionalities into a common software environment like Eclipse. Overall all tools interaction with each other build the tool chain to support the development process.

With TopCased and Rodin there already exist two toolchain which cover the whole software development process by using some of the tools described before. As they use the Eclipse environment other tools can be added to this toolchains. The Why3 toolchain provides a software verification platform for different theorem proofs.

5.3 Summary

Overall a large number of tools is available for different tasks in the software development process. Currently most companies use commercial software, but a growing number of open source software is available. Correspondingly most interviewees reckon that open source software which is proven in use and for which a strong service support can be provided is suitable for safety relevant software development.

The interviews have mainly been focus on modelling tools as for verification and validation usually a combination of specified tools and tool additions is used. Artisan Studio, Atelier B, iLock, SCADE and Simulink have been the tools used by most companies for their software

development. Usually these tools are customised to fit the needs of the individual development process and further tools are added to provide additional functionalities.

6 Conclusions and Recommendations

The existing technologies for software development in railway signalling already provide a wide range of different means of descriptions, methods and tools which can be used to handle the different steps of the software development process as it is required by the EN50128. As shown in chapter **??** usually more than one means of description is needed to capture the different levels of abstraction during the various phases of the development process. To provide the provability which builds the basis for the open proof approach a formal description of all specifications is needed at one point against which the requirements can be verified and validated.

The interviews have shown that formal, model-based development methods are already common practice for the lower abstraction levels of software architecture and design. On the high abstraction levels of software specifications usually semi-formal means of descriptions are used, which do not allow the use of formal proof. Therefore it has to be evaluated to which extend the common formal means of description can be used to model the textual system specifications and whether it is possible to derive the software specifications based on this description.

As it has been shown different methods can be used to get from textual system specifications to source code. As the openETCS concept already intends to use a formal software specifications as a basis for the software generation, methods to handle these development steps have to be chosen. Thereby a method has to be picked how the textual system specifications have to be formalised. This can be done via semi-formal models or by a direct translation.

Corresponding with the primary development methods for the formalisation of system and software specifications, the definition of software design and architecture and source code generation, the methods for verification and validation have to be chosen. According to the open proof concept this should be able to verify certain properties on the different abstraction levels during the software development. Thereby the abstraction level on which a certain verification can be done and the method that can be used depends directly on the kind of property and the degree of formalisation on the abstraction level. Overall a consistent coordination between provided details and for verification needed information has to be established. Correspondingly it has to be defined which formal proofs or test cases can be used to validated all kind of requirements.

As most of the tools currently in use for software development are commercial products, it has to be evaluated closely which open source tools provide the needed functionality and robustness to be incorporated in the software development. As most interviews confirmed that open source tools should be able to handle the needed development functionalities, acceptance of open source tools should not be a general barrier.

Since a wide range of different functionalities has to be supported by tools it is necessary to find a fitting combination of tools which provide which support the selected development methods. As it can be seen at the already existing toolchains Topcased and Rodin the Eclipse framework provides a suitable basis for the tool combination. Correspondingly the interfaces between the different tools and their supported development steps have to be clearly defined to guarantee a safe data exchange.

Overall it can be seen that various means of description, methods and tools are currently used in the signaling industry for software development, but the open proof concept of openETCS needs a more formalised development process as it exists today. Therefore it is necessary to

select a more formalised development process has to be defined which builds on the existing model-based approaches. Based on the development methods probably a combination of suitable means of description for the different abstraction levels have to be chosen from the various means of description used currently. Afterwards suitable tools which support the resulting development process and the needed means of description have to be evaluated and combined to for the integrated toolchain for the software development.

Appendix A: Questionnaire for Interviews

2012-07-19, V1.3

GENERAL INFORMATION CONCERNING THE INTERVIEWS AND THIS QUESTIONAIRE

Attention: This is the version of the questionnaire to be answered by chosen signalling and modelling experts, once it has been agreed on. It is not to be answered by all members of WP 2.

The model of a technical system requires a method, tools and means of description used in a certain context. In the interviews that will be performed in WP 2 of openETCS, the used methods, tools and means of description used by signalling experts in the signalling industry and beyond when modelling complex safety relevant systems will be evaluated. The methods used for formal modelling of complex technical systems will not only be evaluated in the railway domain but in other domains as well. This input from other industries shall enable the openETCS project consortium to benefit from external developments. The evaluation carried out by the interviews will not only focus on the methods used but as well problems experts are dealing with during their daily work with formal modelling and the requested tools, methods and means of description. This will result in setting up requirements on the methods, tools and means of description to be used within openETCS. The results of the interviews will be summarised in a report.

The interview itself is intended to be carried out in a comfortable atmosphere at the premises of the participating signalling experts. The listed questions will only be a guideline, the interviews shall take place as a relaxed conversation. The target of the interviews is to have a good usable and accepted method which can be generally used to model the ETCS system.

A Structure of your organisation

1 Please provide an organization chart of your organisation with focus on the group working on formal modelling.

2 What is the product of your company/ department formal modelling is used for?

3 If a safety relevant system is modelled in your organisation, which parties and/ or groups are involved?

4 Is there a rule for specific methods, tools and means of description to be used in your organisation?

5 What would be required for a change of the methods, tools and means of description?

6 Which requirements have to be fulfilled by your tool and can those be met by open source solutions?

7 If a technical system is modelled, how does the internal approval process work?

8 Have the methods used in your organisation any normative or legal background?

9 Is there a task in your development process which you would like to enhance through formal modelling?

B Development methods

1 Which development method do you use, why?

2 Do your partners use the same development methods?

3 Have the development methods used by your partners an influence on your work?

4 How do you deal with different development methods?

5 How much input for a new project is normally taken from previous projects and in which format are these information usually provided?

C Code development

1 Is your code development model based?

2 Are tests included in your code development?

3 Do you work during your tool development with a versioning system?

4 Do you use a configuration management?

D Verification and Validation of models and code

1 How is your processes structured to verify and validate models and code?

2 How do you create test cases for verification and validation of models and code?

3 Do your partners provide certain process requirements or test cases?

4 Are your partners involved in the whole verification and validation process?

5 How can you adapt your verification and validation process, if different requirements for testing have to be integrated?

E Experience with methods

1 Are you satisfied with the methods you are using?

2 Which problems occur when using the methods?

F Requirement on future methods

1 Which additive functions should modelling methods used in future in your organisation have?

2 Which general requirements do you have on methods?

Appendix B: List Means of description

Table B1. Characterisation with detailed criterias for relevant Means of Description in basis of VDI/VDE 3681:2005

	Criteria														
			Representation			Description of structure			Description of behaviour						
	MoD/ Technique		Formal basis	Textual	Mathematical-symbolic	Graphical	Hierarchical	Composition/ decomposition	Structural change	Deterministic	Non-deterministic	Static	Dynamic	Explicit time representation	No expertise required	Level of standardization	Tool support
ACSL (ANSI/ISO C Specification Language) and C	MoD	o	+				+	+	-	+	-	+	+	-	-	+	+
Ada and Spark	MoD	o	+				+	+	-	+	-	+	+	-	-	+	+
Alloy	MoD	+		+			+	+	o	+	o	+	o	-	-	-	+
CNL (Controlled Natural Language)	MoD	+	+	+			-	-	-	+	-	+	o	-	o	-	o
HOL (High Order Logic)	MoD	+		+			+	+	o	+	+	+	o	+	-	-	o
Lustre / Textual and Graphical Scade	MoD	+	+		+		+	+	+	+	+	+	+	-	-	o	+
OBJ	MoD	+		+			+	+	-	+	-	+	o	-			
RSL (RAISE Specification Language)	MoD	+		+			+	+		+		+	+	+	-	o	+
Timed Petri Nets	MoD	+	o	o	+		+	+	+	+	+	+	+	+	-	+	o

MoD - Means of Description, Tech - Technique

\+ - fulfills criteria completely (can be used) , o - fulfills criteria partially (can be used to some extent), - - does not fulfills criteria (cannot be used)

Overview of means of descriptions - Continued on next page

	Criteria															
			Representation			Description of structure			Description of behaviour							
	MoD/ Technique	Formal basis	Textual	Mathematical-symbolic	Graphical	Hierarchical	Composition/ decomposition	Structural change	Deterministic	Non-deterministic	Static	Dynamic	Explicit time representation	No expertise required	Level of standardization	Tool support
Process Calculi (CCS, CSP, LOToS)	MoD	+		+		-	-	-	o	-	+	-				
State Machines	MoD	+		+	+	o	-	-	o	o	+	+	-	-	-	+
TL (Temporal Logic)	MoD	+		+		-	-	-	+	o	+	+	-	-	-	o
UML 2.0 (Unified Modelling Language) and SysML (System Modelling Language)	MoD	o	+		+	+	+	o	+	o	+	+	+	o	+	+
VDM (Vienna Development Method)	T	+		+		+	+	o	+		+	o	-	-	+	+
Z, B - Method and Event B	T	+		+	o	+	+	o	+	o	+	+	o	-	+	+

MoD - Means of Description, Tech - Technique

+ - fulfills criteria completely (can be used) , o - fulfills criteria partially (can be used to some extent), - - does not fulfills criteria (cannot be used)

Appendix C: Recommended methods

Table C1. Overview of methods and their associated development phases

Method		Trans. of text. specs in formal specs	Trans. of formal requirement specs to formal software	Source code generation	Verification of models and source code	Validation	Creation of documentation	Terminology management/ Intelligent Glossary
Decision Tables		o	o		+	+		
Categorisation of requirements		+	o					
Trusted Components	Library of Trusted/ Verified Components	+	+	+	+	+	o	o
	Tools proven in use	+	+	+	+	+	o	o
	Certified Tools and certified Translators	+	+	+	+	+	+	+
Configuration Management		+	+	+	+	+	+	+
Traceability		+	+	+	+	+	+	+
Model-based development		+	+	+	+	+	+	+
Translation		+	+	+			+	
Structured processes		+	+	+				+
Checklists		+			+	+	+	
Requirement management		+						+
Linked Glossary		+	+				+	+
Terminology Engineering		+					+	+
Functional/Black-box Testing	Prototyping/ Animation	+	+	o	+	+		
	Boundary Value Analysis		+	+	+	+		
	Test Case Catalogue				+	+		
Analysis and Testing	Interface Testing		o	o	+	+		

+ - can be used , o - can be used to some extent

Overview of methods and their associated development phases - Continued on next page

Method		Trans. of text. specs in formal specs	Trans. of formal requirement specs to formal software	Source code generation	Verification of models and source code	Validation	Creation of documentation	Terminology management/ Intelligent Glossary
	Stress-Testing				o	+		
	Test Oracle				+	o		
	Model-based Testing				+	+		
	Performance Testing				+	+		
Error Avoiding Methods	Error Detecting and Correcting Codes		o	o	+			
	Error Guessing		o	o	+			
	Error Seeding		o	o	+			
Modular Approach			+	+	+			
Interpretation			+	+			+	
Dynamic Reconfigurtation			+					
Formal Refinement			+					
Design Analysis	Design Constraint Analysis			o	+	o		
	Design Interface Analysis			o	+	o		
	Design Logic Analysis			o	+	o		
Formal Proof	Theorem Proofing				+	+		
	Modelchecking				+	+		
	Process Simulation	+	+	+	+	+		
+ - can be used , o - can be used to some extent								

Appendix: References

VDI/VDE 3681. Classification and evaluation of description methods in automation and control technology, 2005.

Alessandro Cimatti, Marco Roveri, and Angelo Susi. Etcs requirements specification and validation: the methodology, 2008.

EN50128. Railway applications - communications, signalling and processing systems - software for railway control and protection systems, 2011.

Eckehard Schnieder. *Methoden der Automatisierung: Beschreibungsmittel, Modellkonzepte und Werkzeuge für Automatisierungssysteme: mit 56 Tabellen*. Studium Technik. Vieweg, Braunschweig, 1999. ISBN 3528065664.

Eckehard Schnieder. Integration heterogener modellwelten der automatisierungstechnik. In Manfred Nagl, editor, *Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen*, pages 23–41. Wiley-VCH, Weinheim, 2003. ISBN 9783527277698.

Lars Schnieder. *Formalisierte Terminologien technischer Systeme und ihrer Zuverlässigkeit*. PhD thesis, Technische Universität Braunschweig, Braunschweig, 2010.