Position Paper on openETCS Development Work Flow and Associated Tools The University of Bremen View

Cécile Braunstein Jan Peleska Johannes Feuser

April 23, 2013

Abstract

This document is a proposition from the openETCS team at University of Bremen about the work flow and resulting tool chain architecture definition as required for WP3a. The current contribution considers the "left-hand side of the V-Model", that is, the development-related flow and tools. We plan to investigate the V&V-related work flow and associated tools in the next sprint.

1 General work flow description – Model-Driven Approach

We advocate that the tool chain should rely on the *model-driven* paradigm. The use of domain-specific modelling concepts and an openETCS-specific domain framework is suitable for formal verification and automated code generation, and presents high potential for modelling railway control systems as advocated in [PFH12, HPK11, FP10].

Model-driven work flow. The general notion of model-driven development considers software development as a transformational approach where code is automatically derived from higher-level models. Figure 1 shows the typical model-driven process as characterised by the Object Management Group [OMG11]. The *Meta Metamodel* specifies the rules for defining new modelling formalisms. It has to be sufficiently powerful to introduce language elements, syntactic and static semantic rules which are suitable for the domains – in our case, the ETCS domain – to be modelled.

Using the rules of the meta metamodel, one or more domain-specific modelling formalisms are specified by means of the *metamodel*: the latter comprises all the language elements available and the rules to be observed when developing a concrete model in the specific formalism designed using the meta metamodel. In openETCS one might decide to introduce one or

more formalisms for modelling the EVC (European Vital Controller – on board computer of ETCS trains) and track-side equipment (balises, track elements, interlocking systems, ...). It might even be useful to apply different formalisms for describing the EVC alone, since its functionality comprises many heterogeneous properties, such as odometric components and communication components, brake control and so on.

The availability of several formalisms has the advantage to allow for stronger specialisation per formalism, so that each of these uses a smaller amount of language elements and fewer modelling rules. On the other hand it has the disadvantage that the semantics of interfaces between models elaborated with different formalisms has to be formally described.

A concrete *Model* specifies functional and/or non-functional properties of the system (component) to be developed. A valid model has to observe the rules of the underlying metamodel. Syntactic and static-semantic conformance ensures that *generators* can produce *high-level code* (C,C++, railway-specific programming languages, ...) from the model.

This code is compiled and linked to the *domain framework*, a collection of libraries or pre-defined services which are expected to be required by every (or at least most) applications in the domain. In the openETCS context it has to be discussed, for example, whether it is desirable to specify a *standardised operating system API* streamlined for ETCS, or, at least, for EVC applications: the applications to be modelled and generated will have to be embedded into a run-time environment executing the application code on the target HW platform. If the API of the run-time environment is not available in the domain framework, the API (and its associated behaviour) always has to be modelled in the application models themselves, which would clutter the models in an unacceptable way.

Availability of methods and tools. At each level of the hierarchy different (open source and closed source) tools are already available, supporting (at least partially) formal modelling semantics and formal model-to-model or model-to-text transformations. The list we have provided is not exhaustive but we try to enumerate the ones available and open.

For the meta metamodeling activity, we can choose among MOF [MOF11], Ecore [SBMaJG04] and GOPRR [Kel97]. The metamodel generation may then be done by the EMF eclipse plug-in [SBPM09], and the concrete model may be obtained by a graphical and/or textual tool, here we can also mention the powerful non-open tool MetaEdit+ [KLR96] . Finally code may be generated (X-text/Xpand or OpenArchitectureWare part of the Eclipse-EMF). University of Bremen can also provide code generators, see section 3 for a brief overview of our tool.

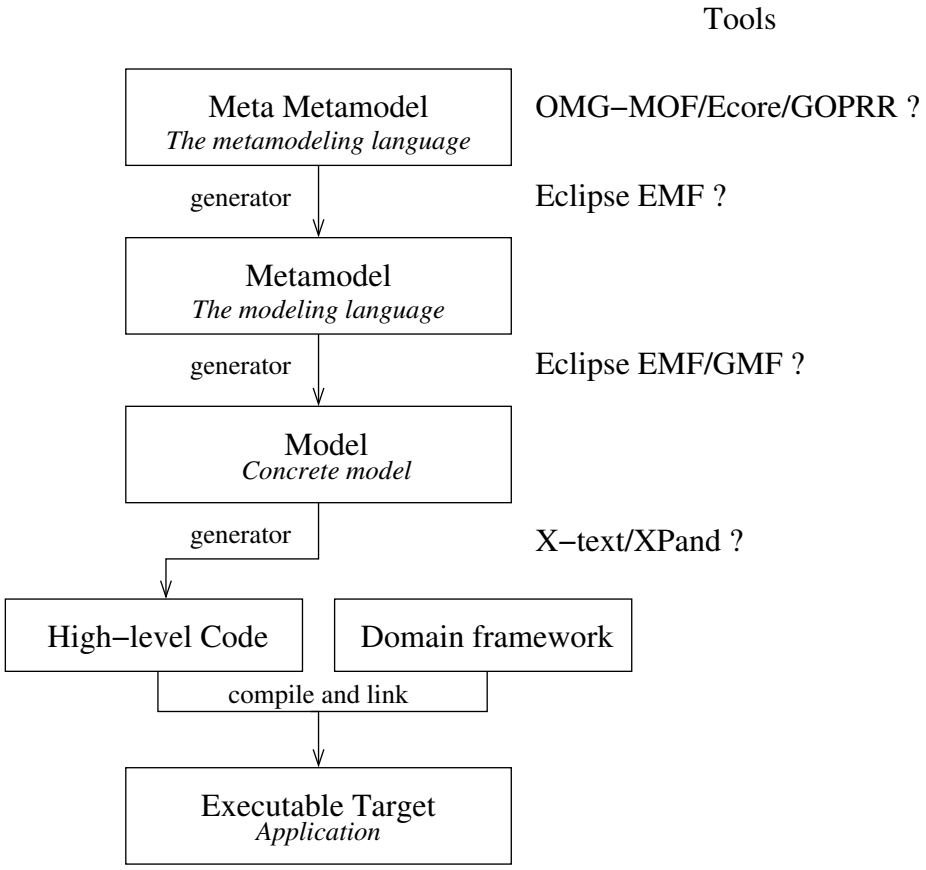

Figure 1: The meta-modelling hierarchy

2 How to choose "good" (meta) meta-models

We have agreed that WP2 should provide some requirements concerning the adequateness of modelling languages. Nevertheless, in the context of model-driven development, one of the tasks of WP3a should be to determine what makes a good candidate for meta-modelling from the tools perspective. We think that we need to define criteria helping to discriminate between the available modelling techniques and associated tools. The criterion should also refer to practical properties such as graphical/text modelling, development costs, availability, open source etc.

We also need to take into consideration the way the produced model will be used. Is our modelling language suitable for code generation in a simulation environment or for a target platform, can we derive test cases and formal verification obligations from the model? The question boils down to:

Which features should the meta meta-model support ?

3 The code generation

University of Bremen already has some experiences with code generators. We can provide a verified transformation from model to intermediate model representation (IMR) and the transformation from IMR to C/C++ code. Figure 2 shows the basic steps of our generator.

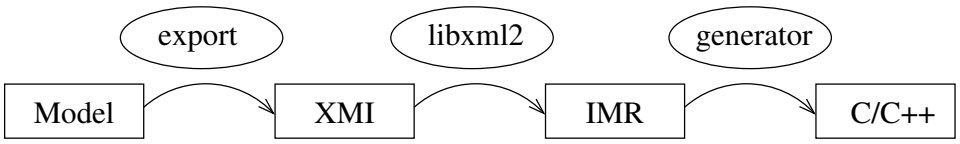

Figure 2: Uni Bremen Generators

Once we obtain the generated code, we need to investigate which Operating system API standard we should use. To ensure the soundness of our approach this decision should be made early in the decision process. If the code generated relies on the availability of specific operating system mechanism, we lost the benefits of the model re-usability. According to our current state of information a candidate for and operating system has already been selected.

4 Certifiable open source – HW standardisation versus virtualisation

A major difference between conventional open source projects and the openETCS approach is that the artefacts resulting from our work flow have to be certified. The availability of (automatically) generated code is most beneficial if the associated V&V artefacts can be used to obtain "immediate" certification. Recalling that certification according to the CENELEC standards EN50128, EN50129, EN50159 [CEN03, CEN01a, CEN01b, CEN01c] requires V&V of the binary code, as integrated on the target hardware, we suggest to discuss two promising options in the openETCS community:

- Operating system standardisation and HW standardisation: one computer architecture for all EVC developments
- Operating system standardisation and virtualisation: one virtual machine running on different HW platforms providing their platform-specific hypervisor

Both options – as explained below in more detail – allow to provide open source down to the binary code, both for application and runtime environment. This would allow to certify the binary code once-and-for-all, since it may run without alterations in every target environment. Only changes in the application or in the runtime environment would require re-certification.

The first option has been realised in the so-called *Integrated Modular Avionics (IMA)* architecture. This approach reduces the development costs and ensures safety and performances requirements by defining a platform with a limited set of re-usable and interoperable hardware and software resources Moreover, the interface between applications and the operating system has been standardised according to the ARINC 653 standard [Aer05].

Another possibility is the use of hardware virtualisation. The advantage of this option is that we may be able to handle different hardware suppliers and different host operating systems via the use of an hypervisor. The hypervisor makes it possible to run guest operating systems simultaneously in virtual machine. It creates an abstraction layer: the guest utilises a predefined hypervisor API providing hardware access [Tan08]. The drawback is then to provide certified hypervisor. Nevertheless, the use of hypervisor for safety-critical system has leaded to an important development of safe-hypervisor over the last years [KEH+09, GPC+03, WJ10]. The openETCS project should benefit from this active research area.

5 Verification and Validation

5.1 V&V Options

The verification and validation part will also benefit from the model driven paradigm, both with respect to higher efficiency and quality.

Functional Testing. Test cases and test data and test procedures may be automatically derived from test models [PVL11, PHL+11].

Verifying Generated Code – V&V of the Generators. Several alternatives exist for verifying code generated from models. The first option is to perform V&V for the generator, so that the artefacts (high-level code or assembler and machine code) created by the generator may be considered a priori as correct, and no additional measures are required to ensure consistency between transformation input and transformation result. Examples for this approach are

- The SCADE tool [Est] is an example of a certified code generator (from SCADE models to C code), whose correctness has been verified by means of comprehensive test suites conforming to the requirements of RTCA DO-178B for generating code of highest criticality in the avionic domain [RTC92].
- The CADUL compiler used by Siemens in railway control systems is an example of a C/C++ compiler which has been validated and approved for application in SIL-4 railway control system developments. It can be used as a compile backend to the SCADE tool for transforming C code generated by SCADE into object code [BPM08].
- Formal compiler verification is an active research field, and many proofs of concept showing that reasonably complex compilers can be formally verified have been given in the literature, we name [Ler09] as one example and refer readers to the literature cited there.

Verifying Generated Code – V&V of the Generation Results. As an alternative, generation results may be automatically tested or even formally verified against their higher-level models:

- The model used as source to the high-level code generation may also be used as a test model. Model-based testing tools are capable today to create comprehensive test suites verifying that the generated code is consistent with the source model. Since these test can also be executed on the target hardware, it is verified at the same time that the machine code integrated on the target executes in a way which is consistent with

the original model. The model-based testing (MBT) approach can be used to create test suites conforming to the highest criticality level of the applicable CENELEC standards, in order to justify that the generated code is consistent to its model [PVL11, PHL+11, LP10].

- The generated result may be formally verified against the model. This formal verification task is easier than proving the correctness of a generator or compiler as a whole, because now just one concrete artefact (the generated code) has to be checked against the transformation source. The theoretical foundations of object code verification, as well as its proof of concept have been established in [PSS98]. In [PFH12, HPK11] these concepts have been refined and applied to the railway domain.

Model-Based Testing of the Generation Results – the Preferred Approach for openETCS. For verifying proper HW/SW integration in systems of highest criticality by means of testing, a large number of tests is required. Therefore this approach was considered as less promising in the past. With the advent of powerful automated test generators, however, we consider this approach to be the most promising and at the same time most cost effective one. Note that this approach requires tool qualification for the model-based testing tool, because otherwise the large number of generated test would have to be manually checked with respect to completeness according to the coverage criteria and with respect to the correctness of PASS/FAIL criteria. This tool qualification has been performed for the RT-Tester tool which we advocate for model-based testing in the openETCS domain [BPS12].

A main advantage of this approach in comparison to performing V&V for generators and compilers is that the latter do not have to be re-verified after improvements and extensions. Therefore we advocate the test-based code verification approach to be applied in openETCS for verifying generated high-level source code or object code of SIL-4 applications. In Figure 3 below, this corresponds to the tool chain fragment

- openETCS Models → Model-Based Tester ← Object Code

In the "Minutes of Meeting – v1.0Draft, Safety Workshop, Work Packages 2,3,4,7, April 2013" by M. Petit-Doche, S. Caillet, S. Pinte, M. Pokam, PF. Jauquet, this approach has been captured in Figure 2 as Safety Process 1. It has not been pointed out there, however, that for the objective of code verification the test model may coincide with the model from where code has been generated: a separate test model is only required if functional tests should verify the correctness and completeness of the code generation model.

5.2 Activities and Tool Chain

Figure 3 summarises the different engines we may need within the development and V&V process.

Description of activities. A model-based approach to openETCS starts with the development of an openETCS model collection. This collection can be classified in several dimensions.

- Structuring according to EVC system components (odometry handler, communication handlers, management of state and level transitions etc.)
- Structuring according to functional (input/output behaviour), structural (architecture) and non-functional properties (for example, safety, security, reliability, availability requirements)
- Structuring according to different levels of abstraction (platform-independent model PIM, platform-specific model PSM)

The models are created within the syntactic and static semantic restrictions of the metamodel and use the informal Subset 026 of the ETCS specification as input.

Software development is based on the model collection. Though in principle, a fully automated model-based development approach could generate object code or even machine code directly from the models, we expect that it will be desirable to have high-level code (C, C++ or similar high-level languages) available since it is easier to trace to the higher-level model than machine code.

The right-hand side of Figure 3 shows one part of V&V activities. Their objective is to verify the consistency between models, high-level code and object/machine code.

- The openETCS model collection will be subject to the peer review of the openETCS community.
- Additionally it is possible to perform complete property checking or incomplete simulations in order to validate completeness and correctness of the models. For identification of properties the conformance test cases specified in Subset 076 may be used, among other possibilities to specify validation properties.

On the left-hand side of the diagram further V&V activities are listed. For model-based system integration testing we propose to develop a second model, the *test model* serving as the bases to model-based testing MBT; this is motivated below in Section 5.3.

As an alternative to property checking and simulation, test suites checking the consistency between model (components) and generated object code can be generated from the input model, as described in Section 5.4 below.

Manual versus automated activties. In principle, all activities identified in the rectangular boxes in Figure 3 may be performed in a manual way. For the ideal approach, however, we recommend that

- code generation,
- compilation,
- property checking, simulation and testing,
- model-based testing,
- equivalence checking

are completely automated, so that only

- openETCS model design,
- model peer review,
- test model design,
- property identification

remain manual activities.

In the following, we indicate the benefits of the model-driven approach in the context of V&V and focus on aspects where the team from University of Bremen may contribute.

5.3 Model-Based System Integration Testing

The objectives of MBT on system integration level are to

- validate the correctness and completeness of the openETCS development model,
- verify that the generated code components cooperate correctly on the target HW, in order to achieve the system-level capabilities.

The first objective implies that the *test model* and the original openETCS development model are separate entities; otherwise the system integration test would just validate that all logical errors still residing in the openETCS development model are really implemented in the code. Even in presence of a formally validated development model, in which high confidence can be placed, we prefer to create a separate test model, because

- the test model may use a higher level of abstraction since only the SUT behaviour visible at the system interfaces is relevant,
- the test model may specify different interfaces to the SUT, depending on the observable interfaces in a test suite; the observation level ranges from black-box (only the "real" SUT system interfaces are visible) to grey-box level (some global variables may be monitored or even manipulated by the testing environment, some task or object communications may be observed etc.),
- the development model may contain errors that are only revealed during HW/SW integration (for example, calculations failing due to inadequate register word size, or deadlines missed due to insufficient CPU resources).

We suggest to create test models on the basis of the ETCS standard (subset 026) and the existing high-level test suites made available in subset 076. The latter test cases should be feasible computations of the test model, so that the test model really creates a *superset* of the existing test suite from subset 076

5.4 Model-Based Component Testing

The second application of MBT is for the objective of code verification. If model-to-text and text-to-text transformations are not formally verified, it is necessary to verify the outcome of each transformation. Since the transformation source is a model M (recall that also high-level code is regarded as a model, represented, for example, by its control flow graphs), MBT suites can be derived automatically from this model to show that the generated code conforms to M.

Observe that in contrast to system-level MBT no redundant model is used for this objective, but the same model M used for code generation can be used: we just have to verify the consistency between code and M, without validating M's correctness and completeness. The latter task is separately performed by means of

- property checking or
- simulation.

5.5 Property Checking

Development models are checked with respect to internal consistency and completeness by means of model checkers capable of property checking.

5.6 Availability of Tools

A wide variety of model checking and MBT tools is available. It has to be noted, however, that in a DSL approach we cannot expect that these tools will be directly able to parse openETCS (sub-)models. Instead, these models will have to be transformed into the formalisms understood by the tools.

- For MBT, the tool RT-Tester (see citations above) accepts different parser front ends to transform models into its own internal representation.
- We plan to establish an openETCS-specifc open source variant based on RT-Tester during the openETCS project.
- For model checkers several input languages are available. It still has to be investigated which tools are most appropriate, and then cross transformation for this tool has to be developed.

References

[Aer05] Aeronautical radio, Inc. Avionics Application Software Interface, Part 1, Required Services, December 2005.

[BPM08] Heike Burghardt, Ralf Pinger, and Stefan Milius. Modellbasierte softwareentwicklung in der bahntechnik. *Elektronik Industrie*, 8/9:24–26, 2008.

[BPS12] J. Brauer, J. Peleska, and U. Schulze. Efficient and Trustworthy Tool Qualification for Model-Based Testing Tools. In *Testing Software and Systems*, Lecture Notes in Computer Science 7641, pages 8–23. Springer, 2012.

[CEN01a] CENELEC. *EN 50128 - Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems.* CENELEC European Committee for Electrotechnical Standardization, Central Secretariat: rue de Stassart 35, B - 1050 Brussels, March 2001.

[CEN01b] CENELEC. *EN 50159-1. Railway applications - Communication, signalling and processing systems Part 1: Safety-related communication in closed transmission systems.* 2001.

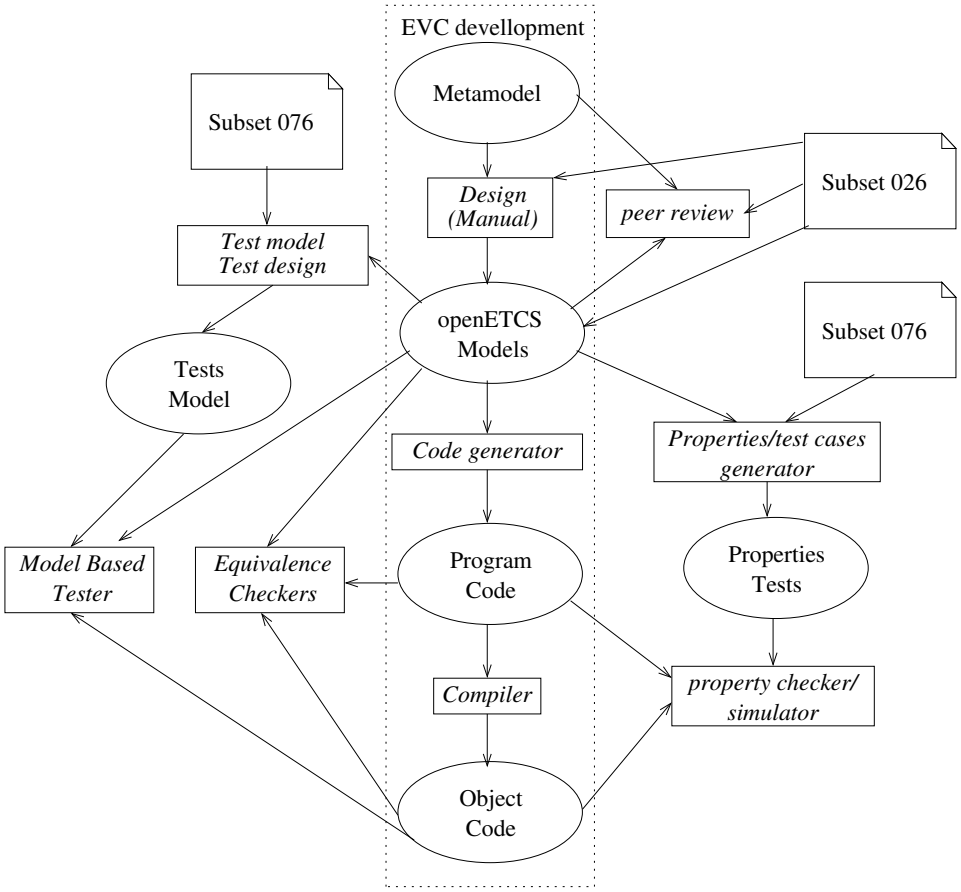

Figure 3: Toolchain with V&V. See Section 5.2 for explanations of activities and artefacts, as well as possible automation steps in a model-based approach.

[CEN01c] CENELEC. *EN 50159-2. Railway applications - Communication, signalling and processing systems Part 2: Safety related communication in open transmission systems.* 2001.

[CEN03] CENELEC. *EN 50129 - Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling.* CENELEC European Committee for Electrotechnical Standardization, Central Secretariat: rue de Stassart 35, B - 1050 Brussels, February 2003.

[Est] Esterel Technologies. *SCADE Suite Product Description.* `http://www.estereltechnologies.com`.

[FP10] J. Feuser and J. Peleska. Security in Open Model Software with Hardware Virtualisation–The Railway Control System Perspective. In *EASST*, volume 33: Foundations and Techniques for Open Source Software Certification, 2010.

[GPC+03] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 193–206, New York, NY, USA, 2003. ACM.

[HPK11] Anne Elisabeth Haxthausen, Jan Peleska, and Sebastian Kinder. A formal approach for the construction and verification of railway control systems. *Formal Asp. Comput.*, 23(2):191–219, 2011.

[KEH+09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kãi Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: formal verification of an os kernel. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220, New York, NY, USA, 2009. ACM.

[Kel97] Steven Kelly. *Towards a Comprehensive MetaCASE and CAME Environment*, chapter Appendix A. Jyväskylä, 1997.

[KLR96] Steven Kelly, Kalle Lyytinen, and Matti Rossi. Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In Panos Constantopoulos, John Mylopoulos, and Yannis Vassiliou, editors, *Advanced Information Systems Engineering*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin / Heidelberg, 1996.

[Ler09] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, July 2009.

[LP10] Helge Löding and Jan Peleska. Timed moore automata: test data generation and model checking. In *Proc. 3rd International Conference on Software Testing, Verification and Validation (ICST'10)*. IEEE Computer Society, 2010.

[MOF11] Omg meta object facility (mof) core specification, August 2011.

[OMG11] OMG. Object mangement group - unified modeling language - UML, infrastructure v2.3, April 2011.

[PFH12] J Peleska, J. Feuser, and A. E. Haxthausen. *Railway Safety, Reliability and Security: Technologies and Systems Engineering*, chapter The Model-Driven openETCS Paradigm for Secure, Safe and Certifiable Train Control Systems, pages 22–52. Information Science Reference, 2012.

[PHL+11] Jan Peleska, Artur Honisch, Florian Lapschies, Helge Löding, Hermann Schmid, Peer Smuda, Elena Vorobev, and Cornelia Zahlten. A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In Burkhart Wolff and Fatiha Zaidi, editors, *Testing Software and Systems. Proceedings of the 23rd IFIP WG 6.1 International Conference, ICTSS 2011*, volume 7019 of *LNCS*, pages 146–161, Heidelberg Dordrecht London New York, November 2011. IFIP WG 6.1, Springer.

[PSS98] A. Pnueli, O. Shtrichman, and M. Siegel. The code validation tool CVT: Automatic verification of a compilation process. *International Journal on Software Tools for Technology Transfer*, 2(2):192–201, 1998.

[PVL11] Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated test case generation with smt-solving and abstract interpretation. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2011.

[RTC92] RTCA,SC-167. *Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178B*. RTCA, 1992.

[SBMaJG04] Dave Steinberg, Frank Budinsky, Ed Merks, and Raymond Ellersick andTimoty J. Grose. *Eclips Modeling Framework*. Addison Wesley, 2004.

[SBPM09] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[Tan08] Andrew S. Tanenbaum. *Modern Operating Systems*. Pearson, 2008.

[WJ10] Zhi Wang and Xuxian Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 380–395, Washington, DC, USA, 2010. IEEE Computer Society.