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

Cécile Braunstein Jan Peleska Johannes Feuser

September 14, 2012

**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 plugin [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.

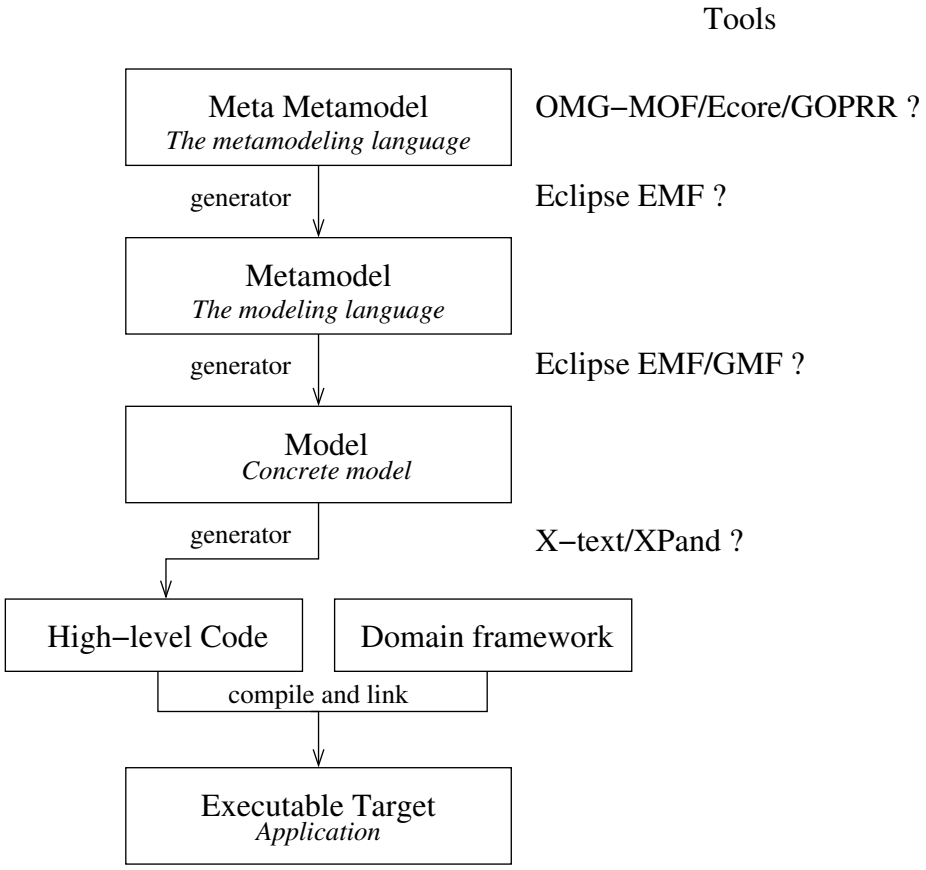


Figure 1: The meta-modelling hierarchy

## 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.

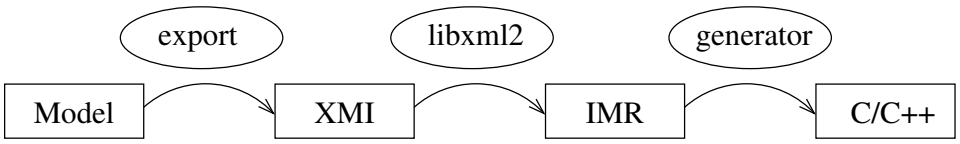


Figure 2: Uni Bremen Generators

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 pre-defined hypervisor API providing hardware access [Tan08].

# References


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

[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.

[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.

[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.

[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.

[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.

[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 Engineer-*

*ing*, chapter The Model-Driven openETCS Paradigm for Secure, Safe and Certifiable Train Control Systems, pages 22–52. 2012.

[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.