Deployment of Model-based Software Development in Safety-related Applications: Challenges and Solutions Scenarios

Mirko Conrad, Heiko Doerr

Research E/E and Information Technology
DaimlerChrysler AG
Alt-Moabit 96A
10559 Berlin, Germany
mirko.conrad@daimlerchrysler.com
heiko.doerr@daimlerchrysler.com

Abstract[1]**:** Since the mid 1990s, model-based development techniques have been adopted for the development of embedded automotive control software. Nowadays, they are also increasingly being deployed in safety-related applications. In usage scenarios such as these, the requirements of standards and guidelines from the safety area have to be adapted and be mapped onto model-based development. This paper discusses the challenges that appear in the process and sketches possible solutions.

1 Motivation and Introduction

Model-based development is becoming the preferred software engineering approach for the development of embedded controls in major vehicle domains, such as powertrain and chassis [CO05]. The basic idea of the model-based approach is that an initial executable graphical model representing the control function to be developed is refined and augmented until is becomes a blueprint for the final implementation from which executable code can be generated automatically. In model-based development, commercial *modeling packages* such as *MATLAB/Simulink/Stateflow* [MSS] and *code generators* based on them such as the RealTime-Workshop/Embedded Coder [RTW-EC] or TargetLink [TL] are applied within the development phases detailed design and coding.

One of the main advantages of this paradigm is that the software development time can be reduced between 20-50% through the use of executable modeling and autocoding, see e.g. [UO04]. Additionally, a faster increase in the maturity level of developed functions is achieved.

---

[1] The underlying research was partly conducted within the scope of the BMBF project IMMOS (01ISC31D); also see www.immos-project.de

These advantages are to be extended to the area of safety-related applications in the future. In such application contexts, additional requirements of safety standards and guidelines have to be adapted. They also have to be mapped onto the model-based development because they mostly date back to before model-based development was introduced.

Building on [DC05] and [CD06], this paper discusses challenges occurring in the process and offers possible solutions based on tangible project experience. Section 2 as a starting point summarizes relevant standards and guidelines. Section 3 describes some of the existing challenges and outlines possible solutions, giving guidance to upcoming research activities. Section 4 concludes the paper.

2 Norms, Standards and Guidelines

The norms, standards, and guidelines listed below can serve as sources of information for the deployment of model-based development techniques in safety-related applications.

ISO TR 15497:2000 Road vehicles - Development guidelines for vehicle-based software: The *Development guidelines for vehicle based software* were compiled by MISRA[2] in the mid-1990s and then transferred to an ISO technical report. They constitute one of the first attempts at standardization in the automotive domain in regards to software development. A revision and an update are under consideration for the time following the completion of work on ISO 26262.

MISRA-C:2004 Guidelines for the use of the C language in critical systems: The *Guidelines for the use of the C language in critical systems*, also compiled by MISRA, describe a subset of the programming language C, which is applicable for the use in safety-critical applications.

SAE J2636 Recommended Practice - C Coding Practice (DRAFT): The SAE[3] Embedded Software Task Force currently collects common automotive *C Coding Practices*. The proposals were developed in order to share lessons learned and best practices for the development of automotive embedded software. They aim to increase the implementation of robust and reliable software and take into account the workshare between automotive OEMs and their suppliers.

IEC 61508:1998 Functional safety of electrical/electronic/programmable electronic safety-related systems: *IEC 61508* is a generic, application-independent standard for electrical/electronic/ programmable electronic safety-related systems (E/E/PES) that is supposed to ease the development of sector-specific norms for E/E/PES. It is applied transitionally in the development of E/E/PES in those areas for which a domain-specific norm does not yet exist. IEC 61508-3 is concerned with the requirements for software

[2] The Motor Industry Software Reliability Association
[3] Society of Automotive Engineers

development.

ISO/WD 26262:2005 Road vehicles - Functional safety: The draft for a sector-specific specialization of IEC 61508 for the automotive domain ('automotive 61508') was compiled by the FAKRA[4] working group 16 Functional Safety. It was handed over as a working draft to the ISO in November 2005 with the objective of standardization. Among other issues, ISO/WD 26262 contains parts dealing with software development (ISO/WD 26262-6) and supporting processes such as the qualification of development tools (ISO/WD 26262-8).

RTCA DO-178B/ED-12B:1992 Software Considerations in Airborne Systems and Equipment Certification: RTCA DO-178B and its European analog EUROCAE ED-12B establish guidelines for the development of software in commercial avionics applications. DO-178B has evolved to become a de-facto standard for the certification of new aeronautical software since its initial appearance.

A successive standard DO 178-C is currently being developed by SC-205/WG-71 and is supposed to be completed in December 2008. One of the topics within the scope of this revision is the consideration of model-based development.

3 Challenges and Solution Scenarios

3.1 Language Description of the Modeling Language

A prerequisite for the deployment of a language within safety-related projects should be a *general language definition regarding syntax and semantics.*

While there are language standards for procedural programming languages such as C, which are open for access (cf. ISO/IEC 9899 [ISO9899]), this is true only to a certain point for the frequently used domain-specific modeling language supported by Simulink/Stateflow. Syntax and semantics of the Simulink/Stateflow modeling language are described extensively in the manuals dealing with the modeling and simulation environment, yet the language description lacks rigor. The semantics of essential language features is complex and often not well-documented, but have often an exemplary character. Moreover, it even remains unclear whether or not the listing of language elements is complete. The user is left alone with finding evidence for ambiguous or error-prone language means. Another negative effect comes from the fact that the semantics of some language parts has been modified throughout the different versions of Simulink/Stateflow. To some extent, the semantics depends on the layout.

There are several approaches in the secondary literature dealing in more detail with the semantics of Simulink and Stateflow [ASK04, CA03, SC04]. As a rule, they do not cover the entire language and are not targeted to the function developer. Rather, they

[4] Fachausschuß Kraftfahrzeuge (Automotive Standards Committee in the German Institute for Standardisation)

serve as the semantic foundation of (analysis) tools based on Simulink/Stateflow. Since an easy-to-deal-with semantics for the overall language, in particular for its state machine part Stateflow, is not to be expected in the near future, a multistage, pragmatic procedure seems to be advisable.

In a first stage, a *complete list of all Simulink basis blocks, Stateflow constructs and connection elements* with their *syntactic variance* (e.g. block parameters) should be compiled. Afterwards their semantics should be specified individually. For the Simulink basis blocks, such a pragmatic description, for example, could be based on the method used by the MSR-MEGMA [MSR02] (see figure 1). However, the variety of block parameters and their combination hampers the semantics description, which, in addition, may depend on overall model properties and simulation parameters. Therefore, this stage of the semantics description is only the first step towards a precise modeling language semantics.

In a second stage the combined use of individual language elements within models and their execution would have to be described. Here, we face the real challenge of the semantics description: the definition of a semantic domain, onto which Simulink/ Stateflow models can be mapped. For Statemate [STM], a purely Statecharts-based modeling tool, such a semantic domain has already been expressed in different places, see e.g. [HN96, LCB00]. But those semantic domains, originally targeted at discrete, event-oriented systems, are not adequate to cover the combined use of Simulink and Stateflow. In the same way, the usage of hybrid automatons (e.g. [AL99]) is not yet mature in order to be sufficient for the Simulink/Stateflow semantics description.

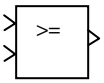
IDxxx	**Relational Operator**
Icon:	
Inputs:	1) u_1: any 2) u_2: any
Outputs:	1) y: any
Init code:	
Run code:	y = (u_1 relop u_2) relop $\in$ {==, ~=, <=, >=, >, <}
Block parameter: 1) Relational operator: == | ~= | <= | >= | > | < 2) Require all inputs to have same data type: Y | N 3) Output data type mode: Boolean | Logical | Specify via dialog 4) …	

Figure 1: Exemplary documentation of Simulink basis blocks

3.2 Safe Language Subset / Modeling Guidelines

During the modeling of safety-related applications constraints need to be observed in comparison to uncritical function parts. Not all of the constructs contained in the

modeling language are suitable for safety-relevant applications. These must be made available to the developer in a convenient form. Initially, based on the complete description of the modeling language used (cf. 3.1), a language subset adequate for safety-relevant applications has to be derived. The aim of such a *Safe Modeling Language Subset* is, on the one hand, to prevent language features from being used in safety-relevant applications, which

- are not defined completely, are ambiguous, depend on the layout or could lead to misunderstandings by the user,
- can be implemented differently by the various code generators,
- ,with a comparable syntax, have varying semantics from the different modeling and simulation tools

and, on the other hand, to provide workarounds in order to avoid known problems in the modeling and simulation tool used.

As a start, the safe language subset can be achieved by restricting the model elements permitted for safety-relevant applications. In practice, this is typically done via *positive lists of permitted language elements*, which are grouped in separate block libraries. Based on this, *safety rules* can be formulated which restrict combinations and settings of language elements.

The rule in figure 2 presents a safety rule on model level, which has been emerged from a similar rule in the MISRA-C language subset. Here, comparisons of floating point variables concerning exact equality / inequality are excluded from the safe modeling language subset in order to avoid the resulting chaotic behavior of the modeled function.

⇨ MC_002:
Absence of equality / inequality tests in floating-point expressions

Category:

☐ Readabilty	☐ Workflow	☐ V&V	☐ Code Gen.	☑ Safety

Automation:
possible

Description:
Floating-point expressions shall not be tested for equality or inequality.
The inherent nature of floating-point types is such that comparisons of equality will often not evaluate to true even when they are expected to.

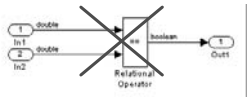


Fig 1-1: Comparison of floating point expressions for equality in Simulink

Figure 2: Definition of a Safe Modeling Language Subset by means of modeling rules

Analogous to a suitable subset of the modeling language for safety-critical applications, a similar restriction can be carried out on the level of the code automatically generated from it. The goals of the definition of a *Safe Implementation Language Subset* are to

avoid possible problems in the implementation language, e.g.

- of C features, which are not defined completely, are ambiguous or have the potential of leading to misunderstandings with the user,
- of C features, which are handled differently by various compilers

as well as the provision of workarounds for known compiler problems.

The exact definition of a safe C language subset depends on how the C code is generated: manually or automatically. For this reason, in addition to the widely accepted MISRA-C rules (cf. section 2.2), a language subset for autocode is currently being developed within the scope of the MISRA autocode forum.

Since presently experiences with language restrictions on the implementation level still prevail, the conformance with the restrictions is currently checked mostly at the C code level. Non-compliance with the rules sometimes leads to long feedback loops due to the process of identifying and turning off the reasons for violations. Consequently, the medium-term goal is to conduct the checks on model level. The code generators' task would then be to ensure that 'safe' Simulink/Stateflow models are compiled into 'safe' C code (figure 3).

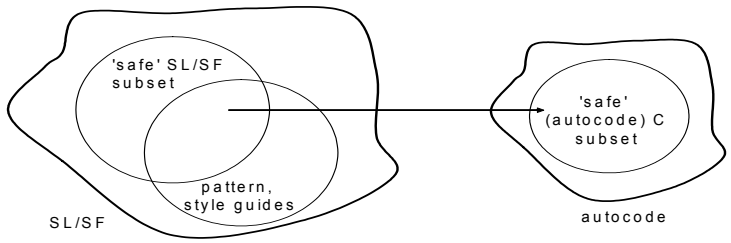


Figure 3: Relationship between 'safe' language subsets on model and code level

3.3 Safety Patterns

Proven methods and techniques which are used within the scope of classic programming in order to ensure safety, can be transferred to the model level and be used in form of so-called *safety patterns for modeling* in a standardized way. Safety Patterns can be regarded as design patterns [GA94] on model level. Know-how in the form of approved model constructs can also be processed in pattern form. Anti-Patterns can represent unsuitable model constructs. So, existing collections of guidelines in the individual organizations can be extended by special safety patterns. Still, benefit and application area must be explained on a case-by-case basis for each pattern.

An example for safety patterns in model-based development is that of fault detection. That feature can be achieved by executing the algorithm once with floating point arithmetics and once with fixed point arithmetics. With suitable configuration of the

code generator it is thus possible to generate different C code fragments, which are executed with the same input values both on a floating point ALU and a fixed point ALU. The results of the two calculations are then compared with each other by an appropriate comparator. The safety pattern depicted in figure 4 explains the principle.

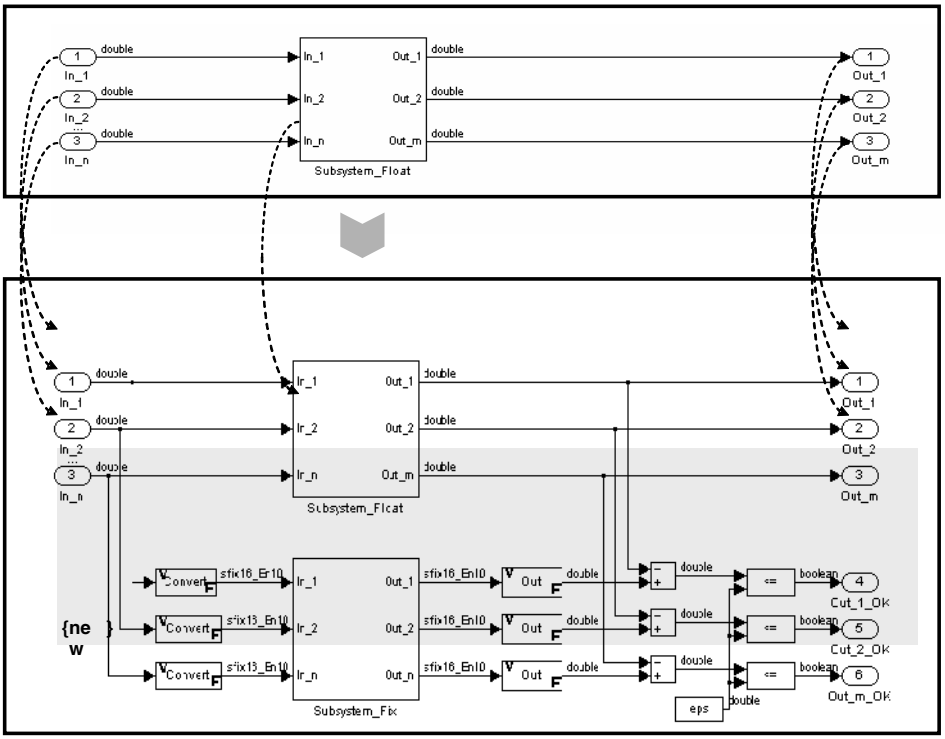


Figure 4: Safety patterns for increasing the fault detection

Positive experiences with web-based front ends for managing and accessing guideline collections are available. An example is the e-Guidelines[5] infrastructure [CFP05]. Compliance with the guidelines can partly be automated with tools such as MINT [MINT] or the Simulink Model Advisor. A linking of guideline data bases and corresponding guideline checkers would be desirable, as would be abstract languages for the description of the rules to be checked.

Further research addresses more complex model analyzers, which, for example, identify 'bad smells' or possible application contexts (host graphs) for positive patterns in models and then interactively enable rule-based model transformations. This way, it would be

[5] www.e-guidelines.org

possible to anchor modern programming approaches on model level, such as the aspect-oriented programming. An example is the above mentioned safety pattern, in which the extra functionality can be supplemented to an existing model.

3.4 Safeguarding the Code Generation

If autocode is deployed in safety-related environments, the quality of the code generators is of paramount importance. At present, code generators are not yet as mature as C or ADA compilers which have been proven in use. For that reason, code generators must be safeguarded as part of the model-based development tool-chain to such an extent that errors possibly incorporated by the code generator can be detected and avoided as far as possible. Safeguarding model-based code generation has become a research topic during the last years, see e.g. [SWC05] for an overview.

A reliable and accepted way to increase confidence in the translation tool such as a code generator is to validate it by means of a test suite. For this reason, a generic *Autocode Validation Suite* (AVS) is now under construction, which is capable of thoroughly validating model-based code generators based on Simulink/Stateflow. Of special importance within such a test suite for code generators is the validation of the optimizations performed by the code generator. A systematic approach to test code generator optimizations can be found in [SC05].

3.5 Standard Compliant Development

Depending on the different (automotive) safety integrity levels, standards such as IEC61508 and ISO/WD 26262 suggest techniques and measures that are supposed to be applied in the individual development phases. Within the scope of a tangible project it is necessary to document which measures were implemented how and/or which replacement measures were employed. The project-specifically accruing effort for this can be reduced effectively if respective generic documents (templates) for typical model-based development projects with the tool chain used in the organization are made available. These templates are then instantiated project-specifically.

Figure 5 shows an exemplary section of a commented IEC61508 measurement table, which contains generic comments on the realization within the scope of model-based development projects in a separate column. A similar procedure is also applicable to the upcoming ISO/WD 26262.

Technique / Measure	SIL1	SIL2	SIL3	SIL4	Comment
…					
Semi-formal methods	R	HR	HR	HR	SL/SF relies on a semi-formal notation for the detailed design of software modules TL relies on a semi-formal notation for the detailed design of software modules
Modular approach	HR	HR	HR	HR	SL supports hierarchical decomposition SL (R14 and higher) allows modularization of models on file level …
Design and coding standards	R	HR	HR	HR	Design (i.e. modeling) standards can be enforced by model reviews Design (i.e. modeling) standards can be partially enforced by MINT …
…					

Figure 5: Implementation of methods and measures of IEC61508

4 Summary

This paper outlined which coordinated measures are necessary to permit the deployment of model-based software development techniques for safety-related automotive applications. Complicated by the utilization of not formally specified modeling languages such as Simulink/Stateflow, a number of informal approaches such as the definition of a safe subset of the modeling language or safety patterns have to be combined. The necessary tool support is partly available. Particularly the utilization of techniques of model transformation will significantly increase compliance with the rules and speed up the necessary verification and validation activities. Applying the measures and techniques mentioned makes it then possible to gain conformity with safety standards.

References

[Al99] R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee. Formal modeling and analysis of hybrid systems: A case study in multirobot coordination, FM'99: Proceedings of the World Congress on Formal Methods, LNCS 1708, 212-232, Springer, 1999.

[ASK04] A. Agrawal, G. Simon, G. Karsai: Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations. ENTCS 109 (2004): 43-56

[Ca03] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis: Translating Discrete-Time Simulink to Lustre. LNCS 2855 (2003): 84-99

[CD06] M. Conrad, H. Dörr: Einsatz von Modell-basierten Entwicklungstechniken in sicherheitsrelevanten Anwendungen: Herausforderungen und Lösungsansätze. Proc. Dagstuhl-Workshop 06022 Modellbasierte Entwicklung eingebetteter Systeme II (MBEES'06), Dagstuhl, Germany, p.7-17 (Informatik Bericht TUBS-2006-01), Jan. 2006

[Co05] M. Conrad, I. Fey, M. Grochtmann, T. Klein: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler. Inform. Forsch. Entwickl. 20(1-2): 3-10, 2005

[CFP05] M. Conrad, I. Fey, H. Pohlheim: eGuidelines – A Tool for Managing Modeling Guidelines. International Automotive Conference, Detroit (US), Jun. 2005

[DC05] H. Dörr, M. Conrad: Modellbasierte Entwicklung sicherheitsrelevanter eingebetteter Systeme. Invited Talk, Dagstuhl-Workshop 05022 Modellbasierte Entwicklung eingebetteter Systeme (MBEES'05), Dagstuhl, Germany, Jan. 2005

[Ga94] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994

[HN96] D. Harel, A. Naamad. The STATEMATE semantics of statecharts: ACM Transactions on Software Engineering and Methodology; 5(4): 293-333, 1996

[LBC00] G. Lüttgen, Michael von der Beeck, and Rance Cleaveland: A Compositional Approach to Statecharts Semantics. Proceedings of the Eighth International Symposium on Foundations of Software Engineering for Twenty-first Century Applications, November 2000, San Diego, CA USA, pages 120–129, 2000.

[MINT] MINT (product information). Ricardo plc, www.ricardo.com/mint, 2006

[MSS] MATLAB / Simulink / Stateflow (product information). The MathWorks Inc, www.mathworks.com/products, 2006

[MSR02] MSR Working group MEGMA: Standardization of library blocks for graphical model exchange. 2001

[RTW-EC] Real-Time Workshop - Embedded Coder (product information). The MathWorks Inc, www.mathworks.com/products, 2006

[SC05] I. Stürmer, M. Conrad: Ein Testverfahren für optimierende Codegeneratoren. Inform. Forsch. Entwickl. 19(4): 213-223, 2005

[Sc04] N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, F. Maraninchi: Formal languages: Defining and translating a "safe" subset of simulink/stateflow into lustre. Proc. 4. ACM Int. Conf. on Embedded Software, Sept. 2004

[TL] TargetLink (product information). dSPACE GmbH, www.dspace.de, 2006

[UO04] T. Ueda, A. Ohata: Trends of Future Powertrain Development and the Evolution of Powertrain Control Systems. Proc. 30th Int. Congress on Transportation Electronics (Convergence 2004), Detroit, USA, pp. 439-449 (SAE Technical Paper #2004-21-0063), Oct. 2004

[ISO9899] ISO/IEC 9899/ Cor 2: 2004 Programming languages – C. International Organization for Standardization, 2004

[STM06] Statemate (product information). I-Logix, www.ilogix.com, 2006

[SWC05] I. Stürmer, D. Weinberg, M. Conrad: Overview of Existing Safeguarding Techniques for Automatically Generated Code. Proc. 2. Int. ICSE Workshop on Software Engineering for Automotive Systems (SEAS'05), St. Louis, USA, May 2005