2008-21-0033

Model-Based Design for Safety-Related Applications

Ines Fey
Safety and Modeling Consultants, Berlin, Germany

Jürgen Müller
Carmeq GmbH, Berlin, Germany

Mirko Conrad
The MathWorks, Inc., Natick, MA, USA

Copyright © 2008 Society of Automotive Engineers, Inc.

ABSTRACT

Production code generation with Model-Based Design has replaced document-based development and manual coding in various automotive domains such as chassis and powertrain. Safety-related applications are increasingly developed using Model-Based Design as well. For these applications, software development and quality assurance activities within Model-Based Design must meet the requirements of the safety standard relevant to the particular domain. For in-vehicle applications, currently this standard is typically IEC 61508.

This paper discusses workflows for developing safety-related application software components and specific requirements with respect to Model-Based Design. Generally, the objectives of IEC 61508-3 influence the entire software development process. However, some activities are of particular importance. This paper is concerned with the following activities exhibiting certain specifics if they are carried out as part of Model-Based Design: traceability between work products, production code generation, dynamic testing, and design for robustness and reliability.

INTRODUCTION

Because of its ability to address software complexity and productivity challenges, Model-Based Design has become the preferred software engineering paradigm for the development of application software components in central automotive domains such as chassis and powertrain.

The core idea is that an initial executable graphical model representing the application software component to be developed serves as the primary representation throughout multiple phases of software development. The executable model is refined and augmented until it becomes a blueprint for the final implementation through production code generation. In addition, executable models can be utilized for various quality assurance activities.

The Simulink product family is a popular tool chain for Model-Based Design. Simulink and Stateflow support graphical modeling with time-based block diagrams and event-based state machines, and Real-Time Workshop Embedded Coder supports embedded code generation.

In the recent past, Model-Based Design with code generation has been successfully employed to produce software for safety-critical applications. Examples include application software components of the electro-mechanical APA steering system [JSB+08] for the Volkswagen Tiguan, an urban SUV. Stringent software development methods and techniques are already required to satisfy customer expectations and ensure the essential quality and reliability of any in-vehicle software. However, given the safety-related nature of some advanced automotive systems, application of techniques above and beyond existing software development practices must be considered for these applications [CDJ+04]. The requirements imposed by safety standards also have to be met, and the objectives and recommendations outlined therein need to be mapped onto Model-Based Design.

The software development activities for a driver assistance system at Carmeq were evaluated to allow rationalizing such a mapping. In practice, the evaluation of this recent project using Model-Based Design led to consolidated findings that became best practice and will be introduced into guidelines for future projects. In this paper the authors combine these project experiences with more general ideas on using Model-Based Design with

Simulink and Stateflow for safety-related automotive applications.

The safety standard currently relevant to automotive in-vehicle applications is IEC 61508. Part 3 of this international standard, IEC 61508-3 [IEC61508-3], is concerned with software development. In IEC 61508, software failures are viewed as the result of faults systematically introduced during software development. In recognition of this, IEC 61508-3 defines requirements and constraints for the software development and quality assurance processes [MK06]. The degree of rigor required in these processes depends on the criticality of the software component within the embedded application and is expressed in terms of safety integrity level (SIL).

The remainder of the paper is organized as follows: First, an overview on Model-Based Design for IEC 61508 is given. Subsequent sections provide details on activities within Model-Based Design that are carried out in the context of safety-related applications: traceability between development artifacts, design and code generation considerations for robustness and reliability, and dynamic testing of models and generated code.

MODEL-BASED DESIGN FOR IEC 61508

IEC 61508, "Functional safety of electrical/electronic/programmable electronic safety-related systems," was developed in the 1990s as a sector-independent safety standard. By constraining the processes used for the development and quality assurance of the software, IEC 61508-3 [IEC61508-3] attempts to reduce the number of faults introduced by the process and increase the number of faults revealed by the process. The standard provides detailed lists of techniques and measures with recommendations (ranging from not recommended to highly recommended) for each SIL.

IEC 61508-3 dates back to 1998, when most software was hand-coded. As a result, it does not cover advanced software development technologies and must be mapped onto the processes and tools used in Model-Based Design.

An analysis of the standard at Carmeq resulted in a number of requirements to be fulfilled that are related to extraordinary challenges in the development process. Within this paper we cover the major aspects, which are from our point of view traceability between development artifacts, modeling and code generation, and dynamic testing.

TRACEABILITY CONSIDERATIONS

In general, tracing dependencies among several development artifacts is crucial to ensure that software development is manageable. Safety standards typically require traceability among software safety requirements, software development artifacts, and test cases. Mapped onto Model-Based Design, these standards require traces beginning at the software safety requirements level, covering architecture and design at the model level, the generated or handwritten code, and test artifacts on different test levels (see Figure 1).

The standard calls for evidence that the safety requirements have been implemented completely and accurately. Therefore, each development phase needs to show how the work products created can be traced to the safety requirements. This means each development phase creates numerous links between work products created in that particular phase and work products created in earlier phases. This finally results in a network of dependencies via explicit or implicit traces that are created throughout development. We define traces as implicit if the link between work products is defined not directly, but via one or more additional work products. This could be, for example, the trace between test cases and model parts (see dashed lines in Figure 1). All the traces have to be maintained throughout the subsequent development process, especially during or after modification of a work product. Because iteration cycles in Model-Based Design are rather short and dynamic, sophisticated trace management is of utmost importance to efficiently fulfill these IEC 61508-3 objectives on traceability.

Models, code artifacts, test cases, and test results are relevant in order to demonstrate that safety requirements have been implemented completely and accurately. To achieve beneficial tracing in the context of Model-Based Design, IEC 61508 includes the following requirements:

- TRA1: Textual safety requirements should be linked to model elements in a fine-grained way.
- TRA2: Textual safety requirements should be linked to dedicated test cases and test results.
- TRA3: Model elements should be connected with dedicated test cases and results either explicitly or implicitly via a certain requirement.
- TRA4: Source code fragments should be linked to model elements and thus to the corresponding tests and requirements (Figure 1).

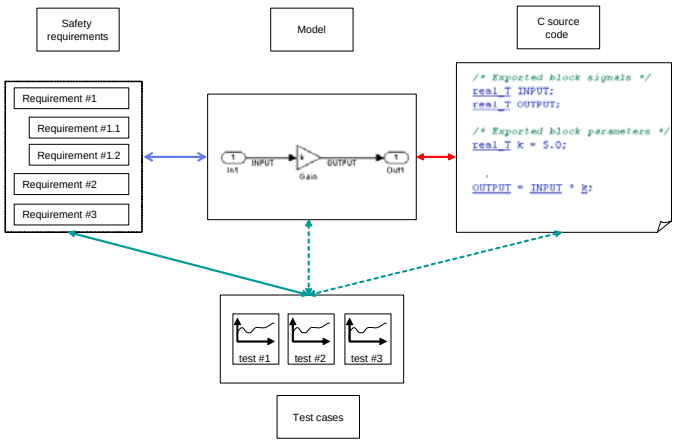

Figure 1. Traceability across artifacts within Model-Based Design.

A requirements coverage analysis following these links could be carried out to get metrics for safety requirements covered by design/code artifacts and test cases as well as to flag uncovered requirements. Naturally, at the end of development, the coverage of safety requirements should reach 100% whatever artifact is in focus.

In practice, the non-safety requirements can be traced using the same mechanism.

The challenge of trace management originates not only from the demand of networked links, but also from the heterogeneous tools and processes used to create the various artifacts.

One way to fulfill the traceability objectives of IEC 61508 is to manually create extensive traceability matrices. But obviously from an efficiency point of view this approach is not reasonable. Since Model-Based Design follows a more dynamic approach than traditional code-based development processes, tool support is highly desirable. A number of trace management tools for a set of tool combinations came into the market not long ago [Reqtify, RMI, ToolNet], but these tools offer only partial support for the day-to-day development work. As a result of its evaluation of these tools, Carmeq has established and maintained in-house solutions for tracing.

The major aspects of the project-specific solution at Carmeq are described in the following sections, which are structured according to the requirements TRA1 to TRA4 of IEC 61508.

TRA1: Textual safety requirements should be linked to model elements in a fine-grained way.

Safety and noncritical requirements are described and managed outside of the model. At Carmeq, Telelogic DOORS [DOORS] is used to gather and maintain textual requirements. Usually, the linking into the model is more or less based on naming conventions. Two types of links can be differentiated here:

- Links between requirements and higher-level subsystems in the model, where the subsystem could be seen as separate functions or architectural components

- Links between signals and model parameters and the interface description within DOORS

One approach used at Carmeq to implement links between model and requirements involves the creation of a surrogate module within DOORS that resembles the model structure. Subsystems in the surrogate module are technically linked to the corresponding requirements in DOORS (Figure 2).

Therefore, the subsystems in the surrogate module are annotated with requirements IDs. Having a naming convention that reinforces the mapping benefits humans working. As an example, a Simulink subsystem would be implicitly linked to a single requirement or a group of requirements in DOORS if both have the same name or some unique common substring. In this case a custom script can analyze both the requirements and the surrogate module and create links if matching names are present.

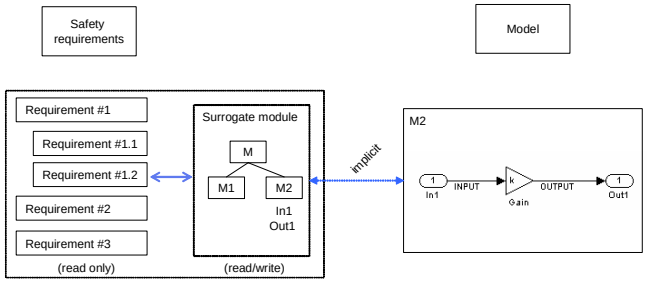

Figure 2. Linking between models and requirements using a surrogate module.

This approach has one major advantage over commercially available tools: This kind of implementation is applicable to situations where the requirements module in DOORS is read-only for modeling and/or test engineers — that is, in situations where a direct link into the requirements wouldn't be feasible.

The basic prerequisite to putting this approach into practice is a structural equivalence between the requirements and the model. In practice, such a structural equivalence down to the level of software units or modules can be ensured only if the teams writing the requirements and the teams creating the models belong to the same or-

ganizational unit. When the development process is divided between an OEM and a supplier, structural uniformity is hard to ensure. However, if the requirements hierarchy and the model structure are consistent, assessing and evaluating the links between requirements and model subsystems can be automated following the defined identifier mapping.

The mapping of information with respect to the interface (e.g., signals, parameters, or configurable parameters) is often implemented by adding attributes to the specification in DOORS. This means that all names used in the model on interface elements are added to the specification. Specialized scripts could be used to either:

- Ensure that the defined signals and parameters are still in use within the model
- Find out about interfaces in the model that are not specified on the level of requirements

TRA2: Textual safety requirements should be linked to dedicated test cases and test results.

Testing activities ranging from model testing to processor-in-the-loop (PIL) testing are planned and executed using specialized tools (see section on dynamic testing). The test execution tool manages all the test cases and the test results emerging from the different test levels. Traces between test cases and relevant requirements, which are additionally required, have to be established via manual links. During test case design all requirement IDs that are expected to be covered by each test are gathered.

The analysis of the achieved requirements coverage can be done using a specific management tool or project-specific Excel templates.

TRA3: Model elements should be connected with dedicated test cases and results either explicitly or implicitly via a certain requirement.

Obviously, this requirement is already fulfilled if TRA1 and TRA2 are satisfied. But it can be achieved only if a trace management tool can visualize these dependencies.

TRA4: Source code fragments should be linked to model elements and thus to the corresponding tests and requirements.

Links between model parts and code fragments are fortunately implemented by the modeling tool and the code generator. During the code generation process, comments are included in the code that enable the modeling engineer as well as the tool itself to switch from a dedicated subsystem to the corresponding source code part and back.

The connection to tests or requirements is naturally available if TRA1 to TRA3 are fulfilled. But, as for TRA3, it can be achieved only if a trace management tool can visualize these dependencies.

In a nutshell, the demands for linking the safety requirements from the specification via the model to implementation and test can be satisfied by using a set of scripts and add-ons to the commercial software development tools. The implemented solutions focus on the explicit linkage between two domains — that is, links between requirements and models, between requirements and test cases, and between models and source code (see the solid lines in Figure 1). Although there is agreement on the necessity and the benefits of tracing the links beginning at the requirements via models to source code and tests, a number of problems need to be addressed in order to achieve IEC 61508 compliant software development. Most important, the support for the overarching trace management and the analysis capabilities for requirements coverage analyses and forward/backward impact analyses must be enhanced.

Furthermore, the demand for sophisticated trace management support results from the number of change cycles in a project using Model-Based Design and from the dedicated needs to maintain the implicit links. Current solutions typically provide link mechanisms but usually no means to separately monitor traces with respect to safety requirements coverage.

Therefore, future development projects have to establish and expand the available trace management tool support to satisfy the increasing traceability requirements resulting from IEC 61508-3.

MODELING AND CODE GENERATION CONSIDERATIONS

One of the essential advantages of Model-Based Design is the possibility of using code generators to translate the algorithms described in Simulink and Stateflow into source code (typically ANSI C) and thereby drastically shorten the implementation phase.

To deploy generated code in safety-critical applications, means for robustness and reliability should already be included at the model level. Utilizing only carefully structured models using a subset of features/blocks for safety-critical systems is recommended.

However, the application domain is characterized by a strong demand for efficient and lean source code that can be satisfied only by applying advanced optimization strategies provided by modern code generators. This constraint has to be considered as well. Safety-critical projects are facing the challenge of balancing these sometimes conflicting needs.

Overseeing the implications of certain modeling or coding constructs as well as balancing the conflicting needs

generally requires experienced project teams with a sound understanding of the relationship between model and code structures.

Carmeq's experiences developing driver assistance functions have shown the following aspects to be highly relevant when using Model-Based Design to create robust and reliable software:

- MCG1: Modeling patterns and best practices that facilitate deployment in safety-related applications should be documented and used.
- MCG2: Control structures and algorithmic parts (calculations) should be separated and never mixed.
- MCG3: Models should be enhanced with checking functionality covering signal ranges and dynamics, default values, and exception and error handling. Measures protecting mathematical operations against overflows should be used carefully.
- MCG4: Data types and casting operations should be used strictly as defined by the utilized modeling and code generating tool.
- MCG5: Comparison operations should be limited to values of the same type.
- MCG6: Interface mechanisms of the generated code should use `get` and `set` functions.
- MCG7: The various suggestions and recommendations should be compiled into a set of modeling guidelines for safety-related applications. If feasible, automatic guideline checking should be provided.

An example that motivates the general need for documenting modeling patterns for safety-related applications is the use of Multiport Switch blocks (Figure 3). Typically the code generator implements a Multiport Switch as a switch-case statement in C. Developers should be aware when using Real-Time Workshop Embedded Coder that the last input will be the "default" case of the generated code. Using this knowledge, the developer is able to define the default behavior of the generated code by connecting according model components to the last inport of the Multiport Switch block.

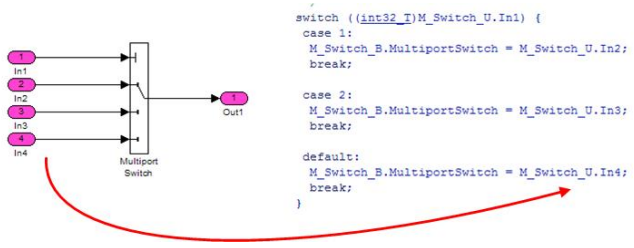

Figure 3. Use of Multiport Switch blocks.

Based on Carmeq's experience, a clear separation of data and control flow aspects was beneficial to achieve designs of manageable complexity that are easy to comprehend and verify. Therefore, at Carmeq, control signals should not be part of calculations, neither in state machines nor within mathematical blocks.

Carmeq considers range checking of input signals to be a "must have." Therefore the data type should be chosen such that the signal range can be checked — that is, the physical range of the data type should be a superset of the specified signal range. Based on signal ranges the protection measures against overflows in math operations should be reviewed on a case-by-case basis to determine if they are necessary. This way unreachable code could be avoided and the code size could be reduced.

Designs with improper or missing typecasts are a common source of error because the code generator may generate code with compiler dependencies. Missing type casts from signed to unsigned data are especially problematic. A better design explicitly specifies the type casts and thus allows the code generator to generate proper type casts that are robust and compiler-independent, as shown in Figure 4.

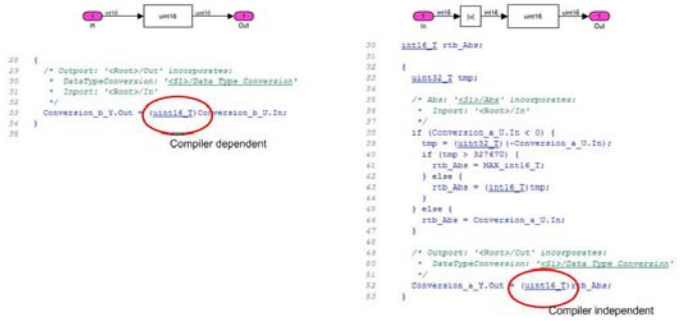

Figure 4. Use of casting operations.

To avoid unintended side effects, comparison operations in models should be limited to expression of the same type. This guideline holds especially for basic data types and fixed-point data types. Floating-point expressions should not be tested for equality or inequality. The latter is the equivalent to MISRA-C:2004 Rule 13.3 [MISRA-C] at the model level.

To ensure a deterministic data transfer, interface mechanisms of the generated code should use `get` and `set` functions. These functions should be provided by the module the data originates from or by the operating system. Real-Time Workshop Embedded Coder supports the concept of inlined S-functions. That is a powerful method for utilizing `get` and `set` functions in the generated code and still being able to model the interface behavior for simulation.

Existing general-purpose modeling guidelines collections such as the MAAB guidelines [MAAB] or the Carmeq guidelines catalog [Car08] typically address safety aspects only to a limited extent. Therefore, the existing modeling guidelines should be enhanced by guidelines and best practices derived from safety considerations such as the ones listed above. Following the spirit of IEC 61508-3 [IEC61508-3] such modeling guidelines for safety-related systems should specify good modeling practice, proscribe non-robust modeling features, and specify procedures for model documentation.

To be effective, the guidelines for safety-related applications should be checked automatically where feasible. Static model analyzers such as the Automotive System Development (ASD) rule checker developed by Carmeq GmbH and Fraunhofer FOKUS [RLK+06] or the Model Advisor from The MathWorks [MA] allow partial automation of verifying a model against the modeling guidelines. A dedicated set of IEC 61508 related modeling standard checks for Model Advisor ships as part of Simulink Verification and Validation. These checks provide suggestions for block settings that help to create standard-compliant Simulink models and to identify modeling issues that impede deployment in safety-related applications or limit traceability (Figure 5).

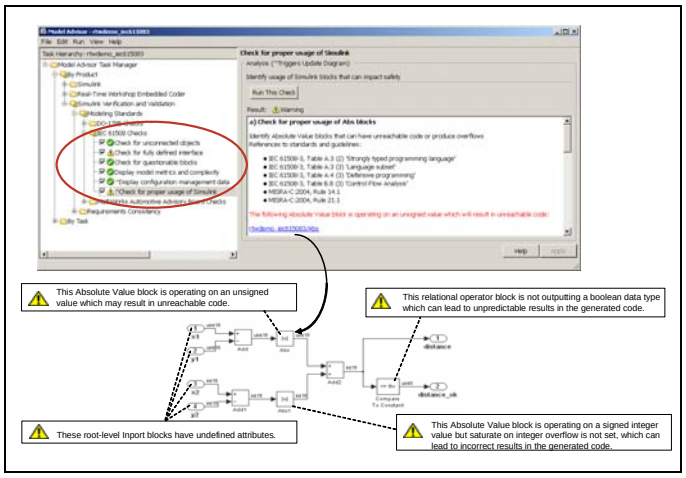

Figure 5. IEC 61508 modeling standard checks in Model Advisor.

IEC 61508 highly recommends that language subsets be used for SIL 3 and higher applications. MISRA-C:2004 is a popular language subset for C code. Ensuring that the generated code complies with coding standards such as MISRA-C:2004 [MISRA-C] and facilitates a high degree of traceability imposes requirements not only on the model design but also on the code generator's configuration. These aspects could also be included in the modeling guidelines and should be checked by corresponding modeling standard checks.

MISRA-C compliance at the code level is of particular importance when generated code and manually crafted code are used in combination.

With TÜV SÜD's recent IEC 61508 (fit for purpose) certification of Real-Time Workshop Embedded Coder [TÜV SÜD], the requirement to use certified translators (IEC 61508-3 clause 7.4.4.3/Table A.3) can be satisfied as well. Using such a certified code generator is expected to ease the certification of generated code.

DYNAMIC TESTING

Dynamic testing is one of the most significant analytical quality assurance techniques for software, as it is the most elementary and certainly the most frequently used form of quality assurance [Lig92]. It is a practical method that adequately takes into consideration the actual development and operating environment of a software system (e.g., code generator, compiler, linker, operating system, target hardware). Furthermore, the dynamic properties of the system (e.g., run-time behavior, computational accuracy of the target system) can be checked. Dynamic testing is the most important and common method used to assure the quality of automotive controls, be they safety-related or noncritical applications. Application-specific testing is required by IEC 61508-3 and other safety standards regardless of the tool chain and the development paradigm used for in-vehicle software development.

Efficient testing requires adapting the testing process to the needs of the overall development process. Thereby, the software development paradigm applied during control software development substantially influences the testing process.

Because the publication of IEC 61508-3 dates back to the 1990s, the standard does not directly provide the required guidance for testing within Model-Based Design. Therefore, the testing-related requirements and recommendations of the standard need to be interpreted and mapped onto Model-Based Design [Con07, EC07, Con08, Con08a].

Testing in Model-Based Design extends the scope known from standard software testing theories. In Model-Based Design, carrying out the largest possible share of the necessary testing at the model level and using executable models as a main source of information for testing seem natural. Utilization of models as test objects permits a distinctly earlier start of test execution compared with code-based development.

Within the scope of Model-Based Design, executable models can be tested and the test information can be reused for subsequent testing of the generated code. Subjecting the software and its preliminary stages (models) to a well-defined combination of testing techniques is essential to detect errors and increase confidence in the correctness and functional safety of the software being

developed. Such test strategies may include combinations of functional and structural testing techniques. The systematic selection of test scenarios from the functional specification and the interface descriptions (requirements-based testing) forms the focal point of the dynamic testing workflow. In addition, an adequate structural coverage requirement can be defined on the model level and used to evaluate the completeness of the test scenarios. If sufficient model coverage has thus been achieved, the test scenarios can be reused for testing the code generated from the model and then embedded within the electronic control unit (ECU) in the framework of comparative equivalence tests (Figure 6). Again a structural coverage requirement, this time at the code level, can be defined and utilized to evaluate the completeness of the test scenarios. This is particularly necessary if model and code structure differ significantly and the chosen model coverage metric does not sufficiently take this into account.

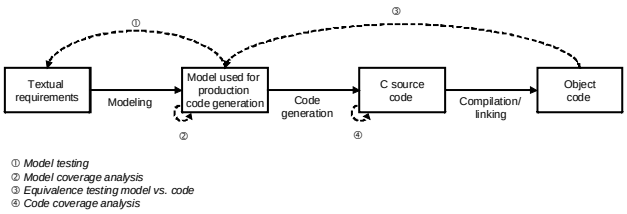

Figure 6. Dynamic testing workflow.

MODEL TESTING

The primary objectives of model-level testing are to demonstrate that:

- TES1: The model components (that is, the model subsystems considered to be a module) perform their intended functions and do not perform unintended functions.

- TES2: The model components interact correctly.

A secondary objective is to ensure that:

- TES3: The model level tests sufficiently cover the different structural parts of the model.

This prerequisite is necessary to ensure the rigor of the subsequent equivalence testing.

The first primary objective can be facilitated by model component testing intended to expose the behavior of the system and to permit a comparison with the specification. Model test vectors should be derived systematically from the software requirements specification according to established criteria (for in-vehicle applications, IEC 61508-3 Table A-5 [IEC61508-3]). See [CF05] for an overview of test design techniques for Simulink models.

After the individual model components have been tested, the software system is assembled and tested according to the integration strategy that prescribes the number of integration levels and the sequence of incorporation of the individual components. During model integration testing, a number of model components representing subsystems and finally the software system as a whole are tested. Software integration testing aims at revealing errors resulting from incorrect component interface implementation, incorrect error handling, or improper control and sequencing of components.

The following model testing capabilities of the Simulink product family can be used to support an IEC 61508-3 conformant module and integration process:

- Signal Builder blocks facilitate graphical test authoring by means of defining test stimuli as piece-wise defined functions. Externally defined stimuli can be imported.

- Predefined Model Verification blocks can be used to check assertions during testing (block outputs should conform to the properties specified within the assertions).

- Simulink Verification and Validation lets end users link textual requirements to test scenarios and verification blocks (see the section in this paper on traceability).

A *model coverage analysis* can be used to assess a given model test suite with regard to its ability to cover structural and data aspects of the model. Structural coverage analyses are well established in imperative programming languages and were made available for Simulink models in 2000. The Model Coverage tool within Simulink Verification and Validation allows conducting Decision, Condition, Modified Condition/Decision, Lookup Table, and Signal Range Coverage metrics based on model tests.

IEC 61508-3 doesn't call out specific coverage goals, but secondary sources such as [SS05] recommend that some test coverage metric should be visible for systems put at SIL 2 and above. At SIL 2 a risk-based approach might be chosen to direct the coverage analysis to components containing complex algorithms and structures and other critical model parts.

If the coverage achieved with the existing test suite is not sufficient, additional test vectors need to be created until the mandated level of model coverage has been achieved.

Carmeq developed an in-house testing environment to integrate the testing process more seamlessly into the overall process of Model-Based Design. The in-house

test environment, named 'm-bedded test suite', has features for defining test scenarios and facilities for test execution (see next section). Some of these capabilities can also be accomplished using Simulink's Signal Builder block, but this block was not available when the m-bedded test suite was first developed.

Engineers at Carmeq are considering using Simulink Design Verifier [SLDV] to aid the creation of additional model test vectors. The test generation feature analyzes the algorithms and logic of the model and generates a set of test vectors covering the various test objectives for the given coverage criterion. Predefined coverage criteria include MC/DC coverage and can be augmented by user-defined objectives.

If full coverage for the selected metric(s) cannot be achieved, the uncovered parts should be assessed and justification for uncovered parts should be provided.

Complete model coverage suggests that the model testing was thorough. Successful, thorough model testing then provides the evidence that the model components behave as specified and interact correctly.

The tested model is used as a "golden reference" for implementation through code generation, compilation, and linking. To verify the code generator/compiler/linker tool chain, the model used as input for code generation and the resulting object code are subjected to equivalence testing (also known as comparative testing or back-to-back testing).

EQUIVALENCE TESTING MODEL VS. CODE

The core objective of *equivalence testing* is to:

- TES4: Validate the model-to-code translation process by demonstrating numerical equivalence between the model and the generated code.

A valid model-to-code translation requires that the execution of the object code exhibit the same observable effects as the simulation of the model for any given set of test vectors. Since complete testing is impossible for complexity reasons, stimuli (test vectors) need to sufficiently cover the different structural entities of the model.

Equivalence testing can be carried out by stimulating both the model used for production code generation and the executable derived from the generated code with identical test vectors. The validity of the translation process (that is, whether or not the semantics of the model have been preserved during code generation, compilation, and linking) is determined by comparing the system reactions (result vectors) of the model and the generated code. In-depth discussions of equivalence testing procedures can be found in [SC03, SC05, SCD+07].

The object code should be tested in an execution environment that corresponds as far as possible with the target environment the code will be deployed to. In practice, the resulting object code is often being executed by means of software-in-the-loop simulation on the host (SIL verification), processor-in-the-loop simulation on the target processor or on a target-like processor (PIL verification), or an instruction set simulator for the target processor (ISS verification). Certification agencies might prefer PIL verification.

As mentioned before, Carmeq uses an in-house testing environment. It handles various kinds of test execution including SIL verification, PIL verification, and equivalence testing [Sch07]. As it is everywhere, model testing is an inherent part of every project using Model-Based Design at Carmeq. However, current development processes for driver assistance applications at Carmeq are based mainly on SIL and PIL verification. Furthermore, an MC/DC coverage metric on code level is integrated seamlessly.

Testing for numerical equivalence is unique in that the expected outputs for the test vectors do not have to be provided [Ald01]. This makes equivalence testing ideally suited to automation.

Simulink Fixed Point offers unique capabilities to carry out a bit-true simulation of fixed-point models. This way a fixed-point model can be completely verified and validated prior to code generation and floating-point to fixed-point conversion issues such don't complicate the equivalence testing.

FURTHER CONSIDERATIONS AND CHALLENGES

Beyond the technical activities, IEC 61508 requires certain supporting activities, including proper test planning and documentation as well as revision control for the artifacts being tested. The test process must be integrated into the overall software life cycle, and corrective actions on failure of tests and detected errors in the software need to be defined.

Currently, assessors have difficulty solely relying on model coverage, even if the interplay between model and code coverage metrics has been examined analytically [Ald01] and statistically [BCS+03]. So on occasion they require additional consideration of code coverage. It is foreseeable that in the future, model coverage will be established as a common technique. Finally, it might be accepted as being equivalent to measuring code coverage. Publishing additional experience regarding model coverage could help to speed up this process.

SUMMARY AND CONCLUSION

Currently, IEC 61508-3 is the relevant safety standard with respect to software development for embedded in-vehicle applications. It defines requirements and constraints for the software development and quality assurance processes. These requirements apply to both Model-Based Design and traditional software develop-

ment. However, implementing these requirements within Model-Based Design requires special consideration and creates specific challenges.

In this paper the authors described workflows and practices for Model-Based Design activities used by Carmeq to fulfill the objectives of IEC 61508-3. These workflows and best practices show the opportunities associated with utilizing the Simulink product family for safety-related embedded software. They are also used to discuss achievements and limitations with respect to tool support currently available and the evolving role of different development artifacts.

The authors anticipate that ongoing and upcoming tool developments will help with fulfilling additional IEC 61508-3 requirements and streamlining development and certification activities. Tool suppliers are expected to advance their tools such that custom tools become less important and the advantages of Model-Based Design can be fully exploited.

REFERENCES

[Ald01] W.J. Aldrich: Using Model Coverage Analysis to Improve the Controls Development Process. Proceedings of AIAA Modeling and Simulation Technologies Conference and Exhibition, Monterey, USA, 2002.

[BCS03] A. Baresel, M. Conrad, S. Sadeghipour, J. Wegener: The Interplay between Model Coverage and Code Coverage. 11. Europ. Int. Conf. on Software Testing, Analysis and Review (EuroSTAR 03), Amsterdam, NL, 2003.

[Car08] Carmeq GmbH: Modellierungsrichtlinien für MATLAB/Simulink/Stateflow, Version 2.0. Volkswagen AG, 2008.

[CDJ+04] B. J. Czerny, J. G. D'Ambrosio, P. O. Jacob, B. T. Murray, P. Sundaram: An Adaptable Software Safety Process for Automotive Safety-Critical Systems. 2004 SAE World Congress, Detroit, MI, USA, 2004 (SAE Technical Paper 2004-01-1666).

[CF05] M. Conrad, I. Fey: Modell-basierter Test von Simulink/Stateflow-Modellen. Proc. TAE Kolloqium Testen im System- und Software-Life-Cycle, Esslingen, Germany, 2005, pp. 278–298.

[Con07] M. Conrad: Using Simulink and Real-Time Workshop Embedded Coder for IEC 61508 Applications. White Paper, Safety Users Group, 2007, www.safetyusersgroup.com/documents/AR070002/EN/AR070002.pdf.

[Con08] M. Conrad: Model-Based Design for IEC 61508 – Towards Translation Validation of Generated Code. Workshop Automotive Software Engineering, Munich, Germany, 2008.

[Con08a] M. Conrad: Model-Based Design for Safety-Critical Automotive Applications. MathWorks Automotive Conference, Stuttgart, Germany, June 2008.

[DOORS] Telelogic, DOORS product information, www.telelogic.de/products/doors/index.cfm.

[EC07] T. Erkkinen, M. Conrad: Safety-Critical Software Development Using Automatic Production Code Generation. SAE World Congress 2007, Detroit, MI, USA, 2007 (SAE Technical Paper 2007-01-1493).

[IEC61508-3] IEC 61508-3:1998. Int. Standard Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements, 1998.

[JSB+08] T. Jablonski, H. Schumann, C. Busse, H. Haussmann, U. Hallmann, D. Dreyer, F. Schöttler: Die neue elektromechanische Lenkung APA-BS. ATZelektronik 01/2008 Vol. 3 (2008) 01, pp. 30–35.

[Lig92] P. Liggesmeyer: Testen, Analysieren und Verifizieren von Software – eine klassifizierende Übersicht der Verfahren. Testen, Analysieren und Verifizieren von Software 1992: 1–25.

[MA] The MathWorks, Inc., information on Model Advisor, www.mathworks.com/products/simverification/description6.html.

[MAAB] MathWorks Automotive Advisory Board: Control Algorithm Modeling Guidelines Using MATLAB, Simulink and Stateflow – Version 2.0, 2007, www.mathworks.com/industries/auto/maab.html.

[MISRA-C] MISRA-C:2004. Guidelines for the use of the C language in critical systems. MIRA, 2004.

[MK06] J. McDermid, T. Kelly: Software in safety critical systems: Achievement and prediction. Nuclear Future, 02(03):140–145, 2006.

[Rau02] A. Rau: Integrated Specification and Documentation of Simulink Models. International Automotive Conference (IAC'02), Stuttgart, Germany, 2002.

[Reqtify] Geensys, Reqtify product information, www.geensys.com/?Outils/Reqtify.

[RLK+06] H. Röbig, A. Leicher, T. Klein, T. Farkas, M. Born, J. Zander-Nowicka: Werkzeugübergreifende Konsistenzsicherung von Artefakten bei der Entwicklung softwarebasierter Systeme im Automobil. Workshop Automotive Software Engineering (ASE'06), Dresden, Germany, 2006.

[RMI] The MathWorks, Inc., Requirements management interface product information, www.mathworks.com/products/simverification/description3.html.

[SC03] I. Stürmer, M. Conrad: Test Suite Design for Code Generation Tools. Proc. 18th IEEE Int. Conf. on Automated Software Engineering (ASE '03), Montreal, Canada, 2003, pp. 286–290.

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

[SCD+07] I. Stürmer, M. Conrad, H. Dörr, P. Pepper: Systematic Testing of Model-Based Code Generators. IEEE Transactions on Software Engineering, Vol. 33, No. 9, Sept. 2007, pp. 622–634.

[Sch07] M. Schmook: Erstellung von Komponenten für eine Entwicklungsumgebung zur modellbasierten Entwicklung von Steuergerätesoftware. Diploma Thesis, FHTW Berlin, 2007.

[SLDV] The MathWorks, Inc., Simulink Design Verifier product information, www.mathworks.com/products/sldesignverifier.

[SS05] D. J. Smith, K. G. L. Simpson: Functional Safety – A straightforward guide to applying IEC 61508 and related standards, 2nd edition. Elsevier Butterworth-Heinemann, 2005.

[ToolNet] Extessy AG, ToolNet product information, www.extessy.com/de/?id=568d4737f4264e9edc6172fd98c0c6ce.

[TÜV SÜD] TÜV SÜD Certificate Database, 193.30.192.53:8080/CertDetail_eng.aspx?CertNo=Z10%2008%2005%2067052%20001&CertTyp=no.

CONTACT

Ines Fey, Senior Consultant, Safety and Modeling Consultants

Ines Fey supports automotive OEMs and suppliers as a consultant in software development projects and process improvement efforts. Her consultant work focuses on the setup and integration of safety activities into the development process compliant with safety norms as well as assistance for activities within Model-Based Design.

She has been working within the model-based development domain for more than 12 years in projects with different OEMs and suppliers. She started her professional career in 1996 as a senior researcher at the Software Technology Lab of DaimlerChrysler Group Research E/E and Information Technology and then worked as a system designer at Carmeq GmbH. Since the beginning of 2008 she has been providing her expert knowledge as an independent consultant.

E-mail: Ines.Fey@samoconsult.com

Jürgen Müller, Business Team Manager, Driver Assistance Systems I, Carmeq GmbH

Jürgen Müller leads Carmeq's Driver Assistance Systems development activities. He started his professional career in the life science industry in 1996. In 2006 he joined Carmeq.

Jürgen Müller holds an M.Sc. in electrical engineering from the University of Rostock. His publication record includes more than 20 papers in various fields and about 10 German and European patents.

E-mail: Juergen.Mueller@carmeq.com

Mirko Conrad, Development Manager, Simulink Certification and Standards, The MathWorks, Inc.

Mirko Conrad manages the MathWorks Simulink Certification and Standards team. Among other responsibilities, he was project manager for the IEC 61508 certification of Real-Time Workshop Embedded Coder. He started his professional career in the automotive industry in 1995 and joined The MathWorks in 2006.

Mirko holds a Ph.D. in engineering (Dr.-Ing.) and an M.Sc. in computer science (Dipl.-Inform.) from Technical University in Berlin, Germany. He is also a visiting lecturer at Humboldt University in Berlin.

His publication record includes more than 60 papers on automotive software engineering and Model-Based Design. He is a member of the Special Interest Group for Automotive Software Engineering in the German Computer Society (GI-ASE).

E-mail: Mirko.Conrad@mathworks.com

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.