

Integration of Requirements into Model-based Development

Mirko Conrad, Ines Fey
DaimlerChrysler AG,
Research E/E and Information Technology
{Mirko.Conrad, Ines.Fey}@DaimlerChrysler.com

Kerstin Buhr
Technical University Berlin,
Institute for Software Engineering and
Theoretical Computer Science
Buhr@cs.tu-berlin.de

Abstract

Executable models are the central artifact in model-based software development. They represent operational descriptions of system behavior and can describe aspects of specification and design as well as of implementation. However, certain types of relevant information, such as abstract top-level or non-functional requirements, are not easily covered within these models and therefore have to be described additionally by textual requirements. Besides these technical reasons, development standards and guidelines also enforce a separate capture of all requirements. Therefore, a dual modeling approach, characterized by the separation of the executable model and its textual requirements, is proposed. Within such an approach, systematic tool-supported linking and tracing between textual requirements and individual model parts is required. To support this we utilize a UML information model that integrates the different engineering disciplines in model-based development. This representation of all the essential information items which occur during model-based development together with their relations was used to identify and select (according to project needs) those links which are useful in this specific development procedure. It was also possible to evaluate the tool support available on the basis of this information model. In this paper we present our experiences gained at DaimlerChrysler Research in using such a dual modeling approach and respective tools.¹

1 Introduction

Nowadays, in modern upper-class cars more than 75% of the vehicle functions are controlled by embedded software. Embedded software thus increasingly determines aspects regarding innovation and the creation of value added to new vehicles [Ro99].

Whereas in the past the complexity and the performance of embedded systems in vehicles was strongly influenced by the technical possibilities, e.g. the microcontroller's performance, now, the development and test process of the control systems and the software embedded therein becomes the limiting factor [Sc98, Be00].

These limitations have been countered, for some years now, by the increased deployment of tool-supported, *model-based development* methods: At the end of the 1990s car makers and suppliers initiated a paradigm shift in the development of software-based vehicle functions from traditional software development on the basis of textual specifications and manual coding to model-based development using executable models and automatic coding [Be00, RCKFD00, MO01, SM01, Ly02, KCFG04].

In model-based development, an executable model of the control software (*functional model*) is already created at an early stage in the development process using block diagrams and extended state machines. This model is initially simulated together with a plant/environment model and can later be directly implemented on the electronic control unit (ECU). The executable model of the software to be realized serves as a basis for all further constructive and analytical development steps.

Within such a model, the structure of the system under development, its decomposition into subsystems (components) and the data and control flow between the

¹ The work described was partially performed as part of the IMMOS project funded by the German Federal Ministry of Education and Research (project ref. 01ISC31D), <http://www.immos-project.de>

different parts are visualized by data flow models (block diagrams). On a lower level the different subsystems can contain either data transforming components or control transforming components. The data transforming blocks, describing continuous or discrete data transformations, are modeled by *block diagrams*. Since the behavior of advanced automotive software depends not only on the current inputs, but also on what has previously and elsewhere happened in the system, transformational blocks alone are not sufficient. These behavioral aspects can be easily represented using extended *state machines* (statecharts). Additionally, C code can be directly integrated into both types of blocks.

The integrated use of data flow models, statecharts and block diagrams (augmented with C code) for modeling control software has proved effective and suitable in an industrial setting. This modeling paradigm is supported by different tools e.g. Simulink / Stateflow (SL/SF) [US02] or ASCET-SD [ASCET01].

Although the executable model of the control software constitutes the central development artifact in model-based development, separately described *requirements* should not be left out of this development process. Functional models represent operational descriptions of system behavior and can describe aspects of specification and design as well as of implementation. However, they lack sufficient means to describe certain types of information, relevant within the development context, such as, for example, abstract, non-operational model requirements, negative requirements or design rationales. Therefore, this kind of information has to be described in addition to the actual model. In industrial practice, the description of requirements largely takes place by means of *textual requirements specifications* [KD03, WW03].

This results in a *dual modeling* approach [CWM98] for ECU software, characterized by the separation of the executable model and its textual requirements (Fig. 1).

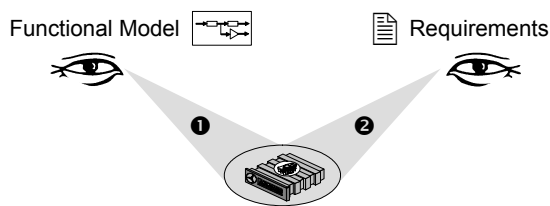


Figure 1 Dual modeling of embedded control software

The explicit separation of the model and its textual requirements is especially helpful for quality assurance. For instance, testing can be facilitated by determining test scenarios according to the specified requirements.

These specified requirements are a sensible foundation for a systematic and effective system or acceptance test for the final system. This is because, in many cases, they form the only document which has been coordinated with the client and serve as the last testing level for the product to be delivered to the customer. Furthermore, it is only the use of requirements which makes it possible to define a functional test of the modeled system independently of its concrete realization (black-box view).

Besides the quality assurance aspect, textual requirements can support the model-based development of innovative, software-based functions, for instance in the following way:

- *Text is a generally understandable means of notation.* In the development team, textual requirements can be used as a discussion platform and to collect thoughts, ideas and variants which, possibly still imprecise and incomplete, can initially be recorded in this form. It is likely that approaches which can only be recorded in a model with a relatively high amount of effort, would otherwise be rejected in this phase.

Under certain circumstances, communication with customers also takes place on the basis of textual requirements. Requests and wishes regarding system functionality are then recorded in a non-technical form which is understandable for the customer, so that the functionality is described using a means of expression with which all participants are familiar. The motivation for following this procedure is, amongst others, provided in [DG95].

- *Models cannot represent all the properties of a system.* The model notations commonly used in current practice are not able to suitably express some essential system properties. These include, in particular, non-functional requirements on performance, the use of resources or quality (e.g. of a control algorithm) as well as so-called negative properties which describe which activities must explicitly not be carried out by the system.
- *Requirements are a key artifact for reuse.* During the iterative further development of a system functionality, e.g. for subsequent product lines, we can assume that abstract, textual requirements tend to be subject to less changes than more detailed solutions actually implemented in models.

Besides these purely contents-motivated rationales for the consideration of information outside of modeling, both the development guidelines, which are the foundation for the development of software-based vehicle functions, as well as other standards also require

a separate gathering and documentation of requirements, see e.g. [DG95].

The use of textual requirements in the way presented above can, however, only be efficient if the requirements themselves are integrated into the entire model-based development process as development artifacts. Primarily, this means that links and

dependencies between the different development elements must be identified and made usable in project work. Therefore, we propose an approach to explicitly identify information items that come into being within the engineering process, and to generally describe interrelationships within an *information model*.

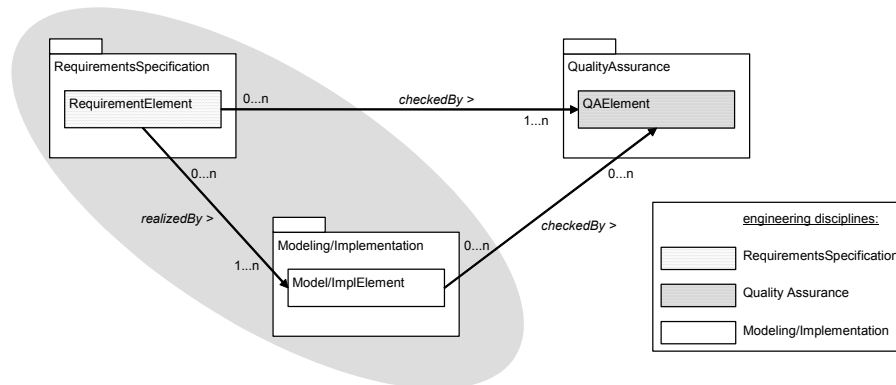


Figure 2 Information model: top-level view

2 An information model integrating different engineering disciplines

We would like to refer to the presentation of the development objects which occur in one or several *engineering disciplines*, as well as their dependencies, orders and hierarchies as an *information model*. Instances of an information model provide a static view of the state, existing at a certain point in time, of a system development².

In the past, information models were mainly examined within the requirements management context [JHNTW99, WW03], and their selective introduction was, in practice, limited to this application context. As model-based processes are increasingly being adopted for the development of control software, it seems reasonable to extend this successful approach to *all* relevant engineering disciplines. Fig. 2 provides a top-level view of the structuring and linking of the main engineering disciplines which are important in model-based development. These are, individually:

RequirementsSpecification, *Modeling / Implementation*, and *QualityAssurance*.

This extension has the effect that all those engineering disciplines are linked to one another within the information model, thus making an analysis of the links between the development objects from different engineering disciplines possible. For our investigation we focused on the relationship between the *RequirementsSpecification* and *Modeling* in particular.

2.1 Tracing between requirements and models

The dual modeling approach outlined previously offers numerous advantages. In order to fully exploit these benefits, systematic tracing between textual requirements and model elements is indispensable. The most evident benefits of an explicit linkage between requirements and models are support for examining complete and adequate realization of requirements by the model, the ability to estimate the effects changes required will have, and support for deriving requirements-based test scenarios for a certain part of the model.

In order to achieve profitable requirement tracing in the context of model-based development, textual requirements must be assigned to individual model parts in a fine-granular manner. While granularity for models

² To be formally correct, one should call the information model an information *meta-model* according to the established terminology of the Object Management Group (OMG) within the context of Model-Driven Architecture (MDA).

is a natural property, the identification of singular, self-contained requirements (e.g. single identifiable sentences, paragraphs or phrases) must be specified methodically in the engineering discipline.

Although there is agreement on the benefits of linking requirements with models, we are faced with a multitude of different approaches for realizing the authoring, management and navigation of relations between requirements and model elements. This is where the advantages of our interdisciplinary information model come into their own. Fig. 3 provides

a detailed illustration of the possible links between requirements elements and model elements, as they can occur in our application context when modeling with Simulink / Stateflow.

On the requirements side, a differentiation is made between requirements, parameter and signal specifications. In the modeling engineering discipline, model elements composed of several elements, such as modules and subsystems are also taken into consideration besides singular elements such as signals, blocks or ports.

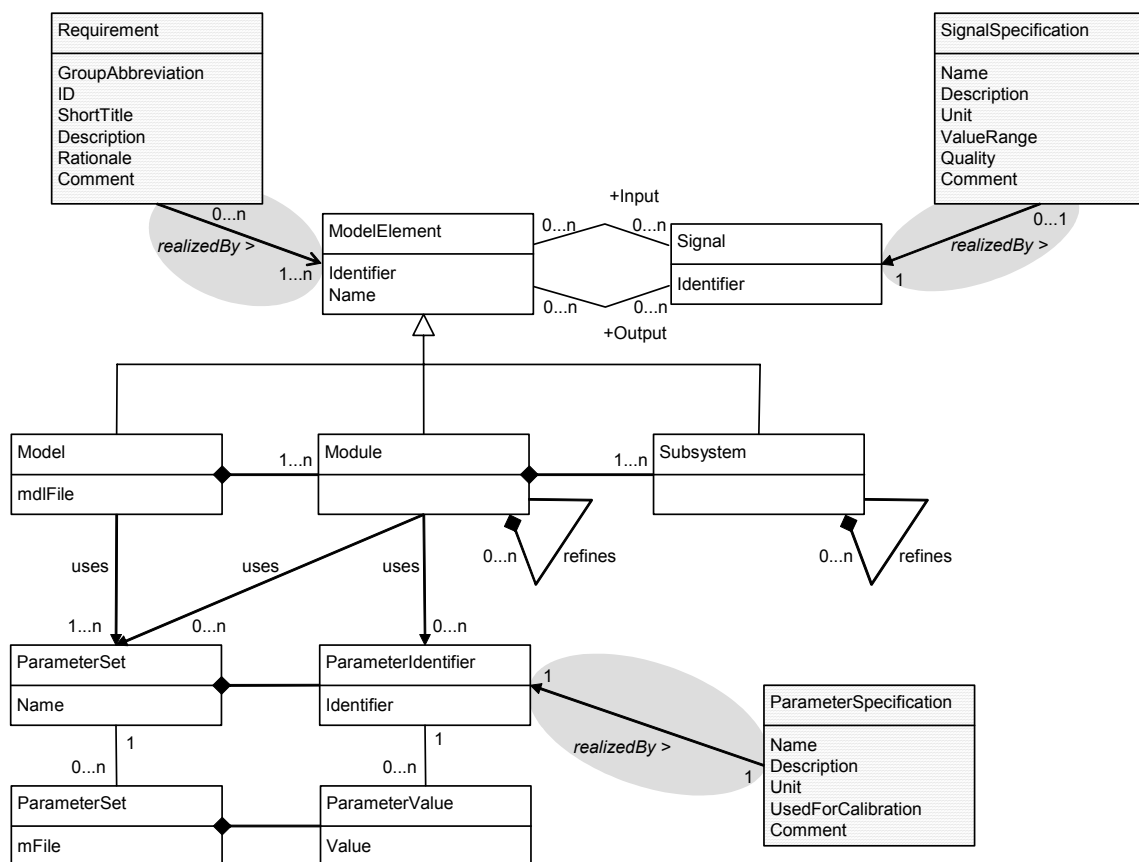


Figure 3 Information model: detailed view

In this way, the information model shows which model elements could principally be linked with the requirements. Based on this representation, those requirements-model-relations which are relevant with regard to traceability aspects, can then be established specifically for each project. A variant which, in our experience, is often used, is highlighted in Fig 3. Here, the focus is on linking requirements with modules

which, in the given context, represent extensive subfunctions. In addition, the signal and parameters specifications from the requirements specification are each linked to signal and parameter identifier objects on the modeling side. The example in the following section should give the reader an impression of how an information model can be instantiated for application in a real-world, automotive development project.

3 Instantiation of the information model

The following illustration uses the development of a chassis control function, the so-called *brake force distribution* (BFD), as an example. BFD is part of today's electronic stability program (ESP).

The central requirement for the BFD is a dynamic distribution of the brake forces, dependent on the

current wheel-load distribution, on the front or rear axle of a passenger car). Here, the situations "forward driving" and "reverse driving" should be considered in particular. The classes Requirement, SignalSpecification, and ParameterSpecification and their attributes filled with values as part of the engineering discipline RequirementsSpecification can be seen in Fig. 4.

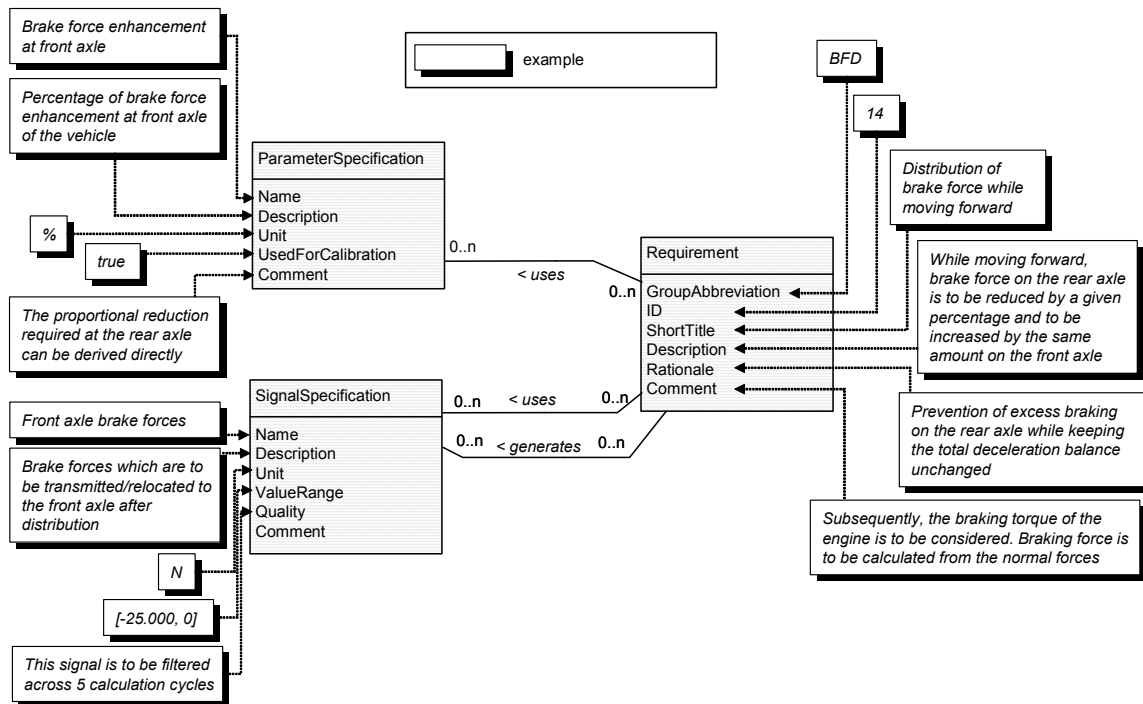


Figure 4 Instantiation: brake force distribution (RequirementsSpecification discipline)

Furthermore, Fig. 4 shows the relations allowed between objects of the class Requirement and objects of the classes ParameterSpecification and Signal Specification. For example, the parameters and signals referred to in a requirement are linked via the relation "uses". In this way, the attributes of the parameters and signals can be reached directly via the relation.

The instantiation of the information model for the engineering discipline Modeling can, as shown in Fig. 5, be performed correspondingly. In the bottom part of the diagram, a section of the BFD function realized in Simulink / Stateflow is shown, which, amongst other things, implements the requirement used beforehand. The relationship of the specific requirement and signal

specification to the specific implementation in the functional model can be easily derived using the information model, as shown in the diagram. The requirement for the distribution of the brake force during forward driving is realized in a Simulink module called `bfd_forw`.

Using the information model we have defined the conceptually desired links between development objects. Nevertheless, the technical representation of development objects, as well as the realization of links between them is not yet supported by tools. The linking concept which is outlined by the information model can, however, be used directly in order to select and evaluate alternatives for tools.

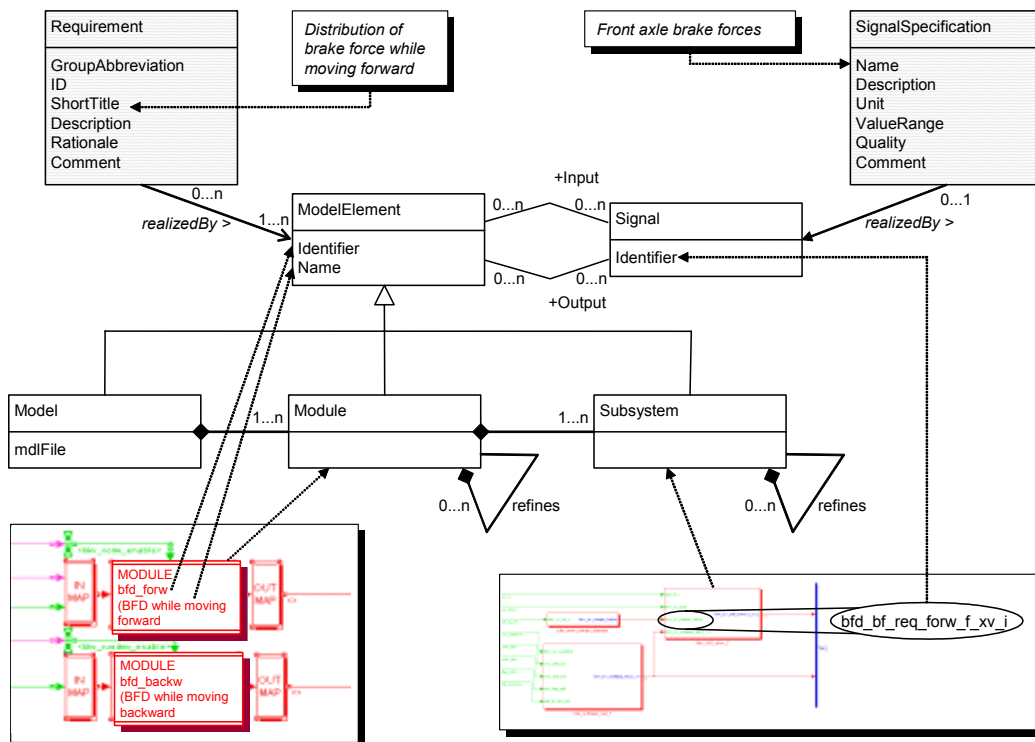


Figure 5 Instantiation: brake force distribution (Modeling discipline)

4 Tool evaluation using the information model

From our experience and from a user's point of view, we can distinguish between three categories of tool support in project work.

- The first category comprises solutions in which requirements and models are managed in separate tools, and the linking of both of the engineering disciplines' reference points occurs informally, e.g. via naming conventions (name matches).
- Within the second category, requirements and models are equally managed in separate tools, however, there is a technical linking of the reference points. This group comprises solutions in which the information from both of the disciplines (RequirementsSpecification and Modeling) is processed and managed in the native tool of the respective application field.
- The third category comprises solutions which extend the tools of one domain with certain functions for the management of information from the other domain. These include, for example, extensions of the modeling tools with specific constructs to annotate

requirements, as is the case, for instance, with the extension of SL/SF with special blocks for requirements management [Ra02].

All three variants can be found in the automotive application development field, depending on the tools currently used for the different engineering disciplines. However, none of these solutions provides tools so far which are able to cover the requirements of model-based development compressed in the information model in a completely satisfactory way with respect to linking and traceability. Against the background of increased prevalence of the use of Doors [DOORS02] within the discipline of RequirementsSpecification in the automotive field in recent years, we will show, in an exemplary way for the second group, the possibility of evaluating a particular tool set in the following. This means, in particular, that a special tool, such as DOORS [DOORS02], is used for the requirements specification, a different tool, such as Simulink/Stateflow (SL/SF) is utilized in the modeling discipline, and that the links are generated and managed within an integration component specialized in the integration with Simulink/Stateflow (R 12.1). The Simulink/Stateflow-Requirements Management Interface (SL/SF RMI) [RMI01] comes under this category.

4.1 Technical linking using native, domain-specific tools and peer-to-peer tool extensions

In the following, an example of the evaluation of a particular tool set is presented, consisting of the following components:

- *DOORS* for requirements management within the specification domain,
- *Simulink / Stateflow* for modeling within the modeling/implementation discipline and the
- *Requirements Management Interface* (RMI) as an integration component

4.1.1 Inner-discipline processing. *DOORS* can deal with all development items in the (textual) discipline of requirement specification, since it is freely configurable³.

In our application context, the use of *Simulink / Stateflow* as a tool for the inner-discipline processing of the development data on functional modeling is a definite choice. For this reason, the information model for functional modeling within system realization was defined according to the syntactical structure of *SL/SF*.

4.1.2 Cross-discipline tracing. The possibility to establish the necessary links in the tool set via RMI will be investigated subsequently:

Ability to refer to linking targets. Not all of the model elements contained in the information model can be linked using RMI, e.g. *Transition*.

Naming and direction of linking: Since model elements are represented within *DOORS* using so-called surrogate modules replicating the block hierarchy of *Simulink / Stateflow*, linking types can be classified by using different link modules (here via the role *realizedBy* stated in the information model). The linking navigation can – as required in the information model – take place bi-directionally.

Cardinality of the linking: The information model allows m:n links, whereby each requirement is to be linked with at least one model element and each model element is to be linked with at least one requirement.

Generally, m:n links can be handled by the RMI. It is not, however, possible to check whether or not all the items have at least one link.

Restriction of the linking targets: In the RMI, linking possibilities cannot be limited depending on the type of source and target. Technically, therefore, it would be possible to also link requirements with *Simulink* signals using the RMI. This, however, is explicitly not admitted in the information model.

Navigation between development objects: Through the direct association between *RequirementElements* and *Model/Implementation-Elements* a very close interrelationship between both items is indicated. Therefore, the ability to directly navigate between them is desirable (cross-discipline navigation) as it is for all information objects within one single discipline (inner-discipline navigation). However, due to the technical realization in RMI by applying surrogate modules for *Simulink / Stateflow* elements within *DOORS*, the user is forced to follow this indirection while navigating between link targets.

4.2 Evaluation results

Considering the development items for each engineering discipline individually, the evaluated tool set meets the required functionality specified in the information model. With respect to the integration of the disciplines, the RMI displays clear weaknesses, particularly with regard to direct navigation between cross-discipline link targets as well as the restriction of the linking possibilities in accordance with our information model (linking targets and cardinalities of links).

Nevertheless, in comparison to other tool sets which we evaluated based on the information model, the tool set considered in this paper currently provides the best conceptual coupling basis. In practical applications, however, the integration component RMI still displays clear weaknesses which – particularly in comparison to the established domain-specific tools *DOORS* and *SL/SF* – are due to the yet insufficient technical maturity of this component. The recently released *SL/SF* version 14 promises further improvements, however, especially where the direct navigation between cross-discipline link targets is concerned.

As a pragmatic solution, the RMI is often replaced in the projects with a manual link between *Doors* and *Simulink* via name match.

For the above application scenario, for instance, this means that functional modules can be found both as headings in the *DOORS* specification module and as a designated subsystem in *Simulink* with the same name. Similarly, signals and parameters, together with their attributes, are defined in *DOORS* modules and signals and named constants with the same name are used in the model. The interface definition of the modules, value range information etc. required for quality assurance measures such as tests and reviews, is taken from these descriptions in *DOORS*. Fig. 6 illustrates this working method.

³ Technically, it is even possible to generate the *DOORS* database structure directly from the information model.

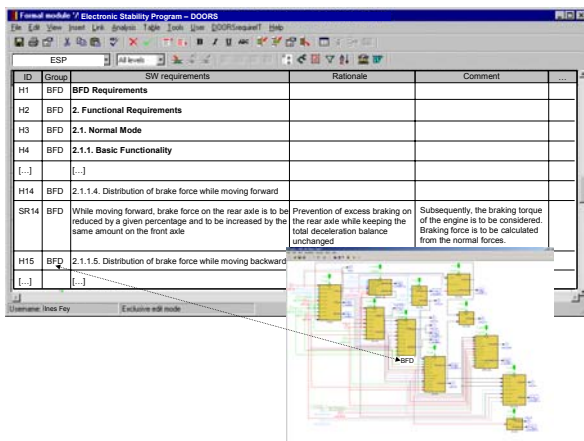


Figure 6 Pragmatic solution for linking requirements and model elements

5 Conclusion and Outlook

In concluding the authors' experiences from their work, one can point out the following:

Model-based development has established itself in industrial practice as a means of dealing with the increasing complexity of embedded control software in the automotive industry. There is a broad range of practically proven advantages in making a functional model the central development artifact.

The use of textual requirements as well as the linking of requirements and model / implementation elements is necessary in model-based development and continues to be of central importance. Besides the executable models of function and environment, separately specified requirements regarding the systematization and comprehensibility of the model-based developed software functions are indispensable.

Model-based development offers broader possibilities for the linking of requirements and development artifacts compared to traditional software development, particularly with respect to a linking of requirements and models. Executable models are available as further development artifacts which can be linked with requirements. The differences in the abstraction levels of requirements and models are, in this case, fewer than those of the requirements and C programs, which facilitates linking. In addition, a greater variety of links to model elements are possible if requirements and models are created in the same organization.

The broader linking possibilities can be taken into account by the extension of information models, which have proven themselves as a methodical basis for these tasks, to the entirety of the model-based development

artifacts. The central engineering disciplines of a model-based development information model are requirements specification, modeling/implementation and quality assurance. These disciplines taken into account, the cross-discipline information model integrates different information or meta models which, until now, were employed independently in the individual engineering disciplines (amongst others, requirements management information model, Simulink/Stateflow meta model).

The UML, as a common formal modeling language for information/meta models, has proven itself for the cross-discipline information model. The meta model for the model/implementation discipline must, in any case, be formal. Since the entire information model should display a uniform level of formalization, the other domains cannot, for logical reasons, be less formal.

One application area for the cross-discipline information model is the investigation of the practical suitability of different tool solutions for the coupling of requirements and models. As a concrete example of its utilization, the evaluation of the coupling of modeling and requirements management tools on the basis of the information model was documented in the present article.

One result of our investigations is that there is currently no clearly preferable tool solution for the technical linking of requirements and models for small to medium projects. Nevertheless, the tool set consisting of DOORS, Simulink / Stateflow and the SL/SF RMI currently provides the best conceptual coupling basis.

These experiences are a quite well-founded base for improving the model-based development of embedded control software in an industrial setting. Although, in this paper, we limited ourselves to parts of the requirement specification and modeling disciplines, there are ongoing investigations in terms of linking of other items, e.g. model elements and test scenarios, based on the same cross-discipline information model. We are confident that the information model-based approach could be generalized and utilized for the systematic evaluation of other tools used to support the model-based development

References

- [ASCET01] ASCET-SD - Development Environment for Embedded Control Systems. White Paper, ETAS GmbH. www.etas.info/download/ascetsd_wp_de.pdf, 2001.
- [Be00] Bechberger, P.: *Modellbasierte Softwareentwicklung für Steuergeräte.* In: *Automobiltechnische Zeitschrift / Motortechnische Zeitschrift Sonderausgabe "Automotive Electronics"*, Friedrich Vieweg & Sohn, Jan. 2000; pp. 16-24.
- [BD03] Buhr, K.; Dörr, H.: *Requirements Driven Quality Assurance.* Proc of 9th Int. Council on Systems Engineering (INCOSE 2003), Washington (USA), 2003.

- [CWM98] Conrad, M.; Weber, M.; Müller, O.: *Towards a Methodology for the Design of Hybrid Systems in Automotive Electronics*. Proc of 31st Int. Symposium on Automotive Technology and Automation (ISATA'98), Düsseldorf, 1998.
- [DG95] *Development Guidelines For Vehicle Based Software*. The Motor Industry Software Reliability Association (MISRA), 2nd Ed., 1995.
- [DOORS02] DOORS, Telelogic AB. www.telelogic.com/products/doorsers, 2002.
- [JHNTW99] John, G.; Hoffmann, M.; Nagel, M.; Thomas, C.; Weber, M.: *Using a Common Information Model as a Methodological Basis for a Tool-supported Requirements Management Process*. Proc. of 9th Int. Symposium of the Int. Council on Systems Engineering (INCOSE'99), Brighton, 1999.
- [KCFG04] Klein, T., Conrad, M., Fey, I., Grochtmann, M.: *Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler*. In: Bernhard Rumpe, Wolfgang Hesse (Eds.): Proc. of Modellierung 2004, Lecture Notes in Informatics (LNI), Vol. P-45, S. 31-41, GI 2004.
- [Ly02] Lygner, M.: *Model-based Development Tool Chain at Volvo Cars*. In: dSPACE NEWS, 1/2002, dSpace Inc., 2002; pp.4-5.
- [MO01] *Model-based Tools Update*. The Hansen Report on Automotive Electronics, Vol. 14, No. 5. www.hansenreport.com, June 2001.
- [Ra02] Rau, A.: *Integrated Specification and Documentation of Simulink Models*. Proc. of Int. Automotive Conference 2002 (IAC'02), Stuttgart, 2002.
- [RCKFD00] Rau, A.; Conrad, M.; Keller, H.; Fey, I.; Dziobek C.: *Integrated Model-based Software Development and Testing with CSD and MTest*. Proc. of Int. Automotive Conference 2000 (IAC '00), Stuttgart, 2000.
- [RMI01] *Requirements Management Interface*, The MathWorks Inc. www.mathworks.com/products/rmi, 2001.
- [Ro99] Rosenstiel, W.: *Eingebettete Systeme. In: Automatisierungstechnik*, Vol. 47, No. 7, 1999; pp. 283-284.
- [SM01] Sax, E.; Müller-Glaser, K.-D.: *A seamless, model-based Design Flow for Embedded Systems in Automotive Applications*. Proc. of 1st Int. Symposium on Automotive Control (ISAC), Shanghai, China, 2001.
- [Sc98] Schmid, H.: *Entwicklung und Realisierung einer Teststrategie mit zugehöriger Testdatenbank für ein Kfz-Elektronik-Testsystem*. Diploma Thesis, Univ. Ulm, 1998.
- [US02] *Using Simulink – Model-based and System-based Design*. The MathWorks Inc. www.mathworks.com, 2002.
- [WW03] Weber, M.; Weisbrod, J.: *Requirements Engineering in Automotive Development - Experiences and Challenges*. In: IEEE Software, Jan/Feb. 2003, pp. 16-24.

