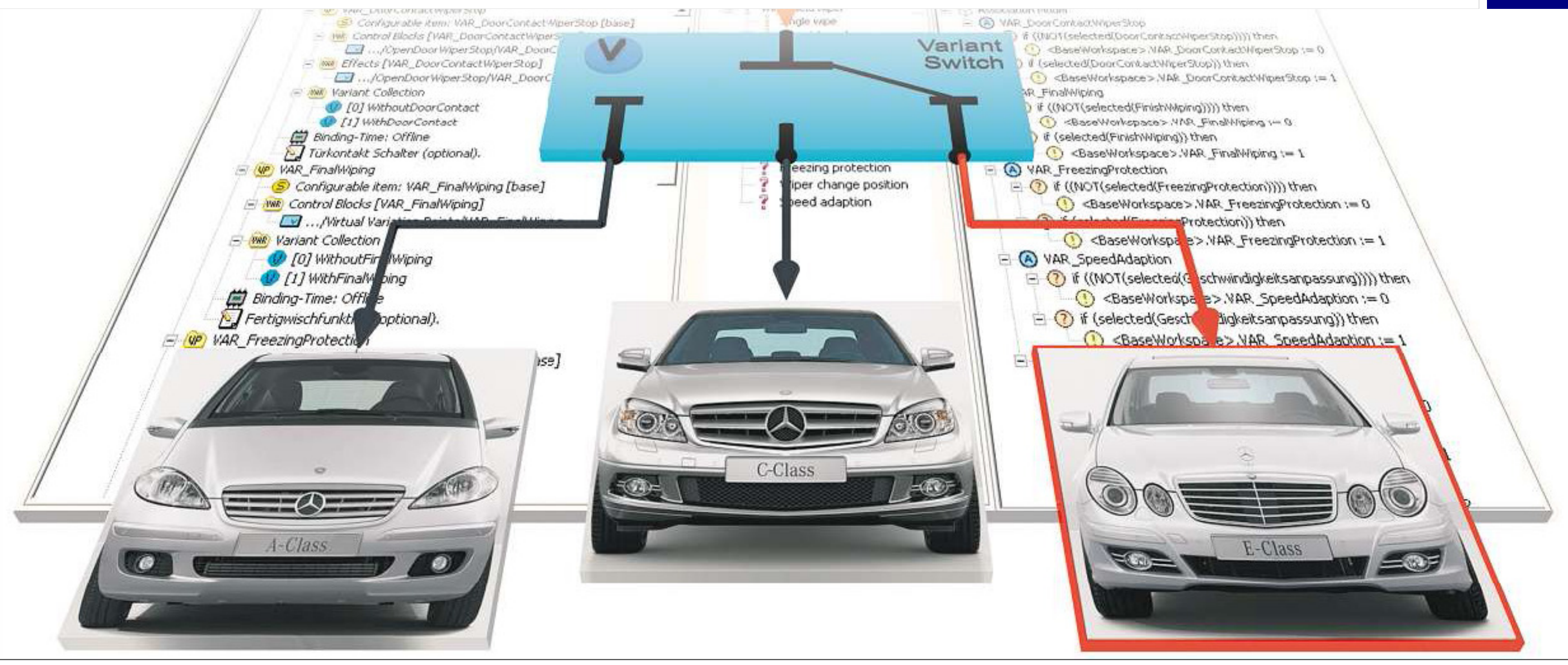

Model Diversity and Variability

Handling functional variants in Simulink models

A characteristic of today's motor vehicles is a wide range of variants with slightly different functions. Since this variability has to be reflected in the software development models, it is essential that there are concepts for systematically handling the variability of functional models. Differentiating between the central and model specific variability information allows uniform handling in Simulink and creates an explicit representation of distributed model variability.

By Christian Dziobek, Joachim Loew, Wojciech Przystas and Jens Weiland

The large number of possible vehicle functions is now a fundamental characteristic of the automotive industry. Of all the Mercedes Benz C class cars manufactured in 2006 the number of possible options is so wide that probably no two were identical. The background is rather complex: different geographical markets have different legislative requirements and customers are given a wide range of choice, starting from body style and engine capacity. More particularly there are the many different technical options that are made possible by electronic systems. As the majority of these options and functions are software-based, embedded software plays a central role in implementing vehicle functions: functional variability in vehicles is automatically based on variability of the software.

The only economical way in which software variability can be handled is through systematic re-use of software artefacts such as architectural descriptions, system specifications, software components, documentation and testing data. This can only be achieved if the structural variability is described in a very systematic way, covering both the variations within specific software artefacts, and the compositional variability, or the way different artefacts are combined.

The automotive sector is increasingly using models and code generation to develop embedded software. Graphical modelling languages, such as the Matlab / Simulink / Stateflow tool chain from The MathWorks, let the designer do specification, modelling and simulation using signal flow and state oriented systems. The result is then turned into executable code using generators such as the Real Time Workshop Embedded Coder from The MathWorks or TargetLink from dSpace.

Simulink models are built by interconnecting elementary blocks and state diagram blocks constructed from signal flow graphs. Complex sub-functions can be encapsulated as subsystems, which in turn can be modelled as signal flow graphs, where the signal connections between blocks represent the data exchanged during simulation of a model. Within Simulink a selection of elementary blocks are available, each for a specific function, such as logical operations or signal routing.

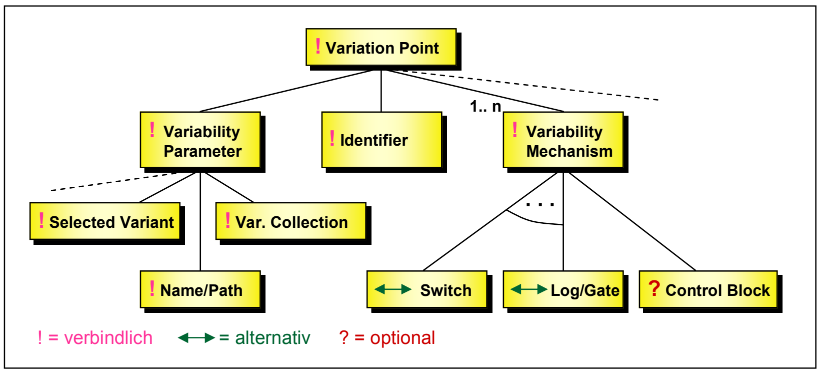

Figure 1: Structure of a Variation Point

Signal flow graphs describe the functional algorithm of a real-time system, primarily based on elementary blocks. But in practice the reusable and configurable functional modules of the basic algorithm are overlaid by the needs of modelling and configuring the different function variants, and Simulink cannot satisfactorily describe these.

This article describes an approach to minimizing these description defects within signal flow graphs created using Simulink / Stateflow. The approach considers, in particular, what information is needed to describe variability within the elementary blocks and how this information can be stored.

Only a systematic consideration of variability provides:

- a uniform description, and subsequently a uniform configuration, of functional variants,
- specific representation of variants in signal flow graphs, distinguishing between standard blocks and variant specific blocks,
- recognition of dependencies between the different functional variants.

The implementation described here is based on Matlab 7.1 and TargetLink 2.1, although it is possible to implement these concepts in just the Matlab tool chain.

Modelling Variability in Simulink

The approach is based on Product Family Engineering [1]. The starting point for describing variability is a Variation Point (**Figure 1**). A Variation Point encapsulates the variability information for each of the specific function variations to be added to the Simulink model. It includes a unique identifier, the Variability Parameter, and the Variability Mechanism.

The Variability Parameter acts as the “tuning knob” of the Simulink model and is configured to select a specific function variant.

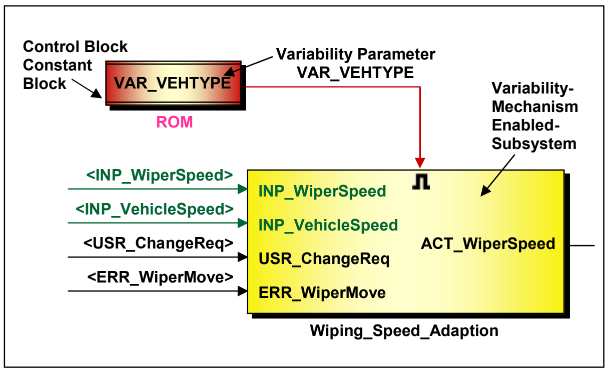

Figure 2: Example of a variation point. Execution is steered via the control block.

It contains a set of values for the collection of variants from which it could be configured: a specific value represents the variant to be selected.

The Variability Mechanism describes how functional variability is implemented in a specific location in the Simulink model, ensuring that a specific variant is executed in accordance with that variant’s configuration parameters.

Simulink’s block library contains a number of blocks which can be invoked to model the variability mechanism. Examples of these are:

- conditionally executable Subsystems,
(Enabled Subsystem, Function Call Subsystem),
- subsystems (IF block, "Switch Case" block),
- "Signal Routing" blocks (Switch Block, "Multiport Switch" block, "Manual Switch” block),
- logic gates, (AND block, OR block).
- configurable Subsystems.

Each of these blocks has its own mechanism to resolve variable functions. The functional range of an encapsulated Enabled Subsystem is activated or deactivated, depending on the value of the Enabled Signal, making an Enabled Subsystem particularly useful for modelling optional functions. In a similar way optional features can be enabled or disabled through the logical AND or OR blocks. A variant can also be selected via the Control Signal of a Switch Block (comparable to a “Switch Case” control structure in C) making the Switch Block particularly suitable for modelling alternative functions.

The majority of these units require an input signal, which steers the execution of the Block. Using a Control Block for selecting a variant is a good option; it can be implemented as Constant Block or “Date Store Read” Block. Depending on the value of the control block, the appropriate variant is executed.

The Configurable Subsystem is an exception: its variant is selected via the value of the Block

Parameter *Block Choice* [2], which contains restrictions for the time of binding, debugging, and for representing the Configurable Subsystem interface.

Figure 2 shows an Enabled Subsystem, where the running process is steered via a Constant Block. The Constant Block can include the selected variant as a value, defining the Variability Parameter by the Block Parameter *Value*.

Variable functions may have effects in several different places within the model, so it is good practice to create a central Variability Parameter in the model- or base- workspace, which can be referred to by the Control Block: in **Figure 2** a central Variability Parameter is VAR_VEHTYPE.

"Separation of Concerns"

Once the information relevant for the modelling of variability in Simulink has been defined, it has to be stored in a way that allows a unified way of handling variability despite the different variability mechanisms. Using a Variation Point to define the variability that exists in the Simulink model and using the Variability Mechanism to resolve this, makes it possible to separate variability information into general and specific elements (**Figure 3**):

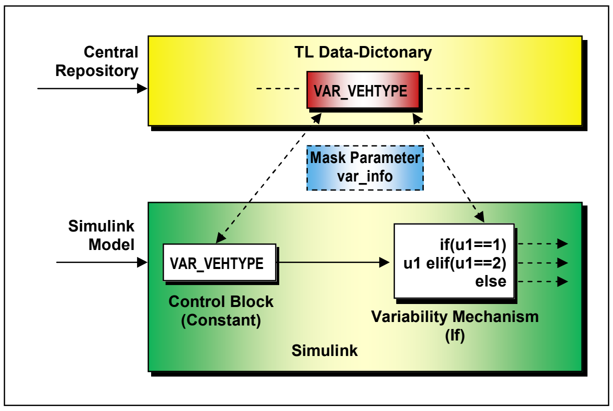

Figure 3: “Separation of Concerns” for Variability Information

- Variability parameters, variant collection and selected variant represent general variability information and can be stored as objects of the same type in a central variant database. Each object has a unique Variation Point identifier which can be referenced from the different control blocks and variability mechanisms of the model; in this example, data is modelled as an object in the TargetLink Data Dictionary. Alternatively, it can be instantiated as a Matlab Structure, a Simulink Data Class, or as an encapsulated Java Class, and as a parameter in the base model or workspace.

- The variability mechanism and the associated control block provide the specific elements. Blocks which express the variability of the Simulink model are defined as variant specific by an added mask parameter VAR_INFO, whose value points to the corresponding object in the central variant database. Using a mask parameter means that a unique identifier is allocated for variability information, since block parameters, such

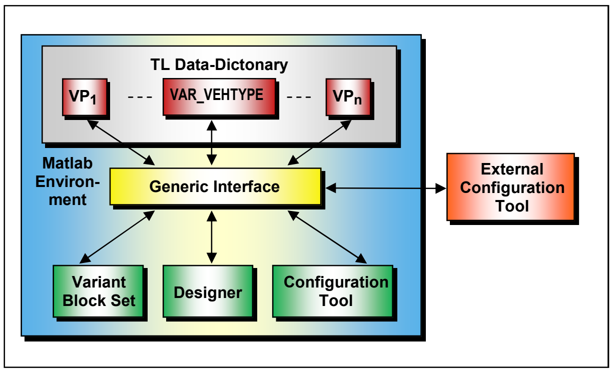

Figure 4: Generic interface for transparent access to Variability information

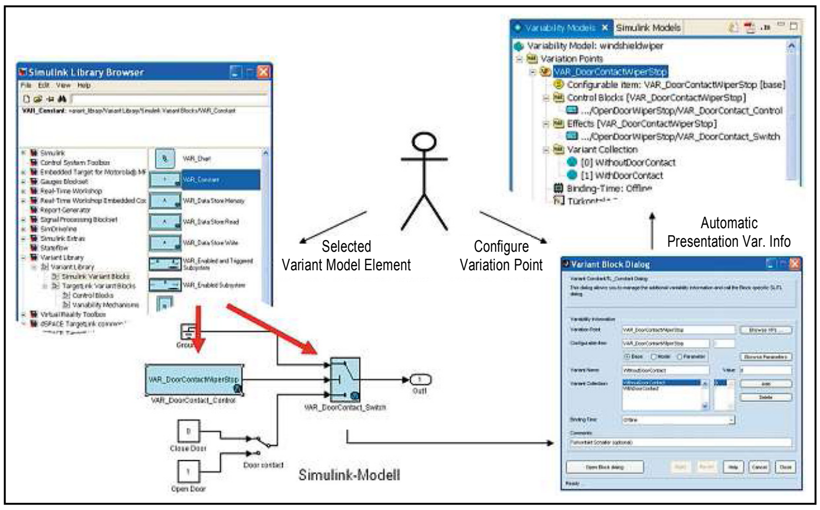

Figure 5: Design of variable functions based on variant block sets

as Tag or Description, could already be used for other purposes.

Partitioning the variability information into a common and a specific part provides uniform access to variability information through a generic interface to the central variant database (**Figure 4**). With this interface, different blocks with their own mechanism for the resolution of variable functions can be used as a Variability Mechanism using an identical Variation Point structure. The designer can have transparent access to the variability in a Simulink model via Matlab functions or via configuration tools.

As part of developing the systematic approach to modelling variability, a separate Variant Block set was designed. It includes specially annotated general Simulink blocks of the library with a "Callback" parameter *OpenFcn* that allows for blocks to be configured using a special variant specific dialogue.

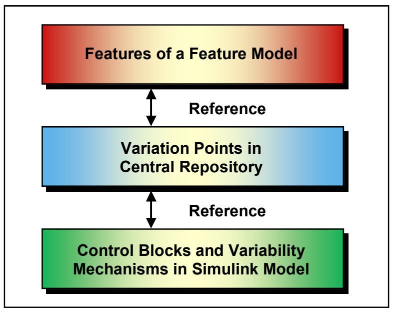
Features of a Feature Model
Reference
Variation Points in Central Repository
Reference
Control Blocks and Variability Mechanisms in Simulink Model

Figure 6: Levels of Abstraction from Feature to Simulink Model

Model-based Development Process

An example will show how variability modelling using the Variant Block sets can work (**Figure 5**). Initially, variant specific blocks are inserted into the Simulink model (**Figure 5**, bottom left). These are later configured using the Variant Specific Dialogue (**Figure 5**, bottom right). This is called using the "Open Callback" function *Open-Fcn* and allows existing Variation Points stored in the variant database to be edited or new Variation Points to be created. When establishing a new Variation Point, the Variability Parameter is defined by:

- a name and path,
- values that the parameter can take,
- the selected variant.

The Variant Specific Dialogue also provides a way to input the block specific information when using TargetLink blocks.

The Simulink model can be viewed as tree structure (**Figure 5**, top right), which can be populated automatically from the available information and provides an explicit representation of variability. By selecting a variant specific block, the corresponding block in the Simulink model is directly called up.

This transparent access to variability in a Simulink model offers a wide range of options for configuration tools. Those tools using Feature Models form an abstract view on functional variants, independent of the Simulink model (**Figure 6**). Features specify domain-specific common and variable concepts from a user's view. (In **Figure 7** the optional selection

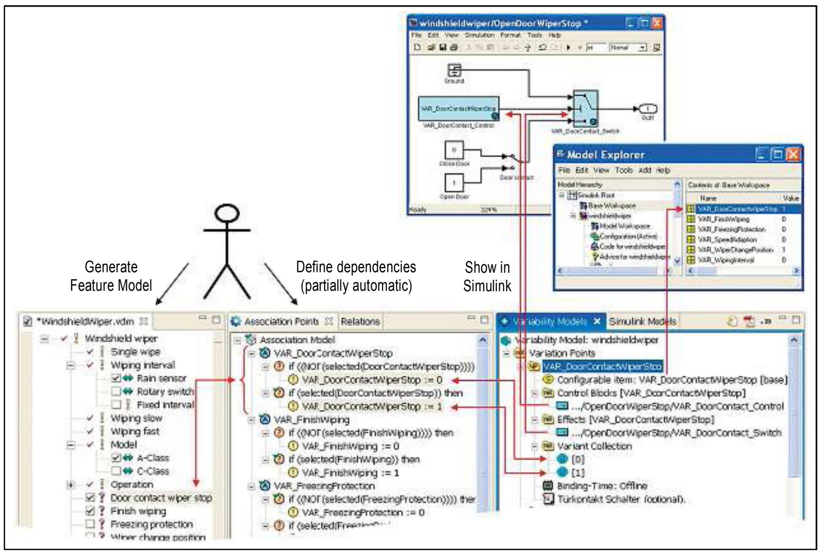

Figure 7: Configuration of Variability in Simulink Models via Feature Models using Door Contact information

of a door-contact function stops the wiper when opening a front door). The result is a comprehensive model of these concepts including their dependencies of model variants (**Figure 7**, bottom left), which distinguishes between necessary, alternative and optional features, as well as (1..n):m group relations.

The relationships between the features of a Feature Model and Variation Points of a Simulink model and how they can be defined (in **Figure 7**, bottom middle, named as Association Model) are described in Reference [3]. With these relationships it is possible to assign the value of a variant specific parameter in a Simulink model, to a particular feature of the Feature Model: pure::variants, from pure-systems, was integrated with the Simulink models to implement this approach [4].

In summary, this approach will help to increase the quality of the model based software to help it cope with significant feature variation.

- by defining dependencies, a selective search for distributed variability in a Simulink model becomes possible, retrieving both the feature which influences the Variability Point of the Simulink model and those features a Variability Point depends upon.
- it is possible to automatically generate valid configurations for a Simulink model from the Feature Model, e.g. from a parameter set or a configuration instruction.
- coupling Simulink models with feature models makes it possible to search for contradictions in configurations of functional models with complex variabilities.
- variability in Simulink models is explicitly visible. All of the variabilities are represented by variant specific blocks in the Simulink model and a common variant database, where all of the variabilities of a Simulink model are centrally managed.

The concepts presented have been developed as part of a research project and are currently in production use for a Mercedes car development. A special Variant Block set, an API based on Matlab functions, and a configuration tool have been specifically implemented to achieve this. hs/*ms*

Literature: Original

[1] Czarnecki, K.; Eisenecker. U.: Generative Programming – Methods, Tools, and Applications. Addison-Wesley, Boston, MA, 2000.

[2] Weiland, J.; Richter, E.: Konfigurationsmanagement variantenreicher Simulink-Modelle. A.B. Cremers et.al. (Hrsg.): INFORMATIK 2005 – Informatik LIVE!. Band 2. Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI). 19. – 22. September 2005. Kölln Verlag, Bonn, 2005.

[3] Klengel, K.; Weiland, J.: Merkmalbasierte Konfiguration variantenreicher Simulink- Modelle. H. Dörr, T. Klein: Unterlagen zum Workshop „Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen“, Modellierung 2006. 22. – 24. März 2006. Innsbruck, Österreich, 2006.

[4] pure-systems GmbH, pure::variants. Eclipse Plugin User Guide, 2007.

Dipl.-Ing.
Christian Dziobek

studied Electrical Engineering at RWTH Aachen. He has worked at Daimler AG since 1998, specifically on the introduction of methods and tools for model-based function development at department E/E for series car development.
christian.dziobek@daimler.com

Dipl.-Inf.
Wojciech Przytas

studied Computer Science and Computer Linguistics at Stuttgart University. He became a doctoral student at Daimler-AG-Research in 2007 and works mainly on variant configuration of model-based embedded software.
wojciech.przystas@daimler.com

Dipl.-Ing.
Joachim Loew

studied Aerospace Engineering at Stuttgart University. He has worked at Daimler AG since 1998. His responsibilities include the development of model-based function design tools for ECUs.
joachim.c.loew@daimler.com

Dipl.-Inf.
Jens Weiland

studied Computer Science at Bundeswehr University in Munich. Since 2000 he has headed research projects at Daimler-AG Research. His focal point is variant configuration of model-based embedded software.
jens.weiland@daimler.com