A Custom Approach for Variability Management in Automotive Applications

Fabian Kliemannel, Georg Rock
PROSTEP IMP GmbH
Dolivostr. 11, 64293 Darmstadt, Germany
{fabian.kliemannel, georg.rock}@prostep.com

Stefan Mann
Fraunhofer Institute for Software and Systems Engineering ISST
Steinplatz 2, 10623 Berlin, Germany
stefan.mann@isst.fraunhofer.de

Abstract—Product lines are receiving more and more attention in software and mechanical engineering processes in the automotive industry. The benefits of using a product line approach are clearly defined in literature and widely accepted. Nevertheless, until now there are only few applications that exploit the variability knowledge during a complete development process. In this paper we argue that all the necessary prerequisites for the handling of variability are present. What is missing is on the one hand an integration of these techniques within the typical area specific development tools (as for example MATLAB/Simulink) and on the other hand the seamless transition of variant information between different engineering tools and development levels.

Index Terms—product line engineering; variability management; architecture descriptions; MATLAB/Simulink; automotive software

I. INTRODUCTION

Product lines are receiving more and more attention in software and mechanical engineering processes in the automotive industry. The benefits of using a product line approach are clearly defined in literature and widely accepted. Nevertheless, until now there are only few applications that exploit the variability knowledge during a complete development process; from requirements engineering to the specification and execution of tests. We argue that all the necessary prerequisites for the handling or management of variability are present, as industrial tools like pure::variants [2] or the Feature Modeling Plugin [20] demonstrate. What we think that is missing is on the one hand a complete integration of these techniques within typical domain-specific development tools, and on the other hand the seamless transfer of variant information between different engineering tools. We previously addressed this problem by defining a reference process that allows for such an integrated approach [9], [10]. While we focused in this work on the early phases of system development as there are requirements engineering and architectural design, we want to extend this approach with respect to the implementation phase now. In this paper, we describe an exemplified integration of variability management within the domain specific engineering tool MATLAB/Simulink [14]. The extension of MATLAB/Simulink was executed as close as possible to the usual development steps within Simulink. Thus, the user is not forced to be trained on a completely new tool. The additional functionality concerning the handling of variability as there are consistency checks, interactive configuration, dead feature detection, and switching between different variants of the Simulink model can be done without using an external variability management tool.

The rest of the paper focuses on the realization of this work, and thus constitutes a proof of concept for the before demanded complete integration of variability management within domain specific development tools.

II. REALISATION OF MODEL-BASED VARIABILITY MANAGEMENT

MATLAB/Simulink is a tool for modeling, simulating and analyzing multi-domain dynamic systems. The main part of Simulink is a graphical editing language based on block libraries which are used to specify dynamic systems. There are standard library blocks which can be used to model variability: enabled subsystems, configurable subsystems and model variants, for example. This was already described by the authors of [3], [7]. But the main purpose of these mechanisms is to model the behavior of single systems. Thus, we extend and reinterpret them to explicitly represent variation points in Simulink. The resulting Simulink extension, which we called "v.control.mbd", consists of

- an additional block library for modeling variation points in Simulink models,
- a consistent data management, and
- a graphical user interface (GUI) for the configuration, analysis and assessment of variant-rich Simulink models.

Our block library consists of three types of variation points (Figure 1): There is an optional subsystem block, i.e. a subsystem which can be present in a configuration, but it does not have to be present. It can be turned on and off during a configuration step. The second type, an XOR variation point block, represents a subsystem which encapsulates subsystem alternatives. If the XOR variation point itself is present in a configuration, then exactly one of its children will be present. The third possibility is a combination of both mechanisms, i.e. an XOR variation point itself can be additionally optional. These introduced variability mechanisms can be used within a hierarchical structure as usual within Simulink. Thus, it is
possible to specify an XOR variation point that again consists of XOR variation points, for example.

In the implementation of v.control.mbd, optional subsystems are realized using enabled subsystem blocks of Simulink connected with a control block (e.g. normalCB in Figure 6). An optional subsystem can be activated and deactivated by changing the respective value of its control block. XOR variation points are realized by standard subsystem blocks. Each subsystem alternative is again realized by an enabled subsystem block connected with a control block. It is ensured by the v.control.mbd implementation that only one alternative is active at a specific point of time.

![Image of v.control.mbd block library in Simulink](Image)

Figure 1. Elements of the v.control.mbd block library in Simulink.

The introduced variation points can be easily used within any Simulink model. The user simply has to insert a block from the v.control.mbd block library. The tool automatically creates the respective constraints and the needed variability data. This data is administered using a structure called VarInfo (see Figure 2). It contains the complete variability data of the current Simulink model and is stored in the base workspace of MATLAB. This approach allows to distinguish between model elements for the product line structure and model elements used to specify the functional behavior.

![Image of VarInfo structure](Image)

Figure 2. Data model of v.control.mbd in Simulink.

The third part of the v.control.mbd plugin is the configuration and analysis user interface shown in Figure 5. The configuration GUI is used to interactively specify a configuration of the Simulink model. A formal analysis engine is used during the configuration step in order to justify all the user made decisions and to check whether all the constraints are fulfilled. Thus, the user is guided during the configuration process to guarantee that the resulting configuration is consistent and respects all the automatically generated constraints [12]. Additional constraints can as well be added by the user in the GUI.

The following section presents a small example to show how the presented prototype is applied.

### III. EXAMPLE APPLICATION OF THE V.CONTROL.MBD PLUGIN

To illustrate the before introduced concepts, a traffic lights product line is taken as an example. It consists of signalers for cars and for pedestrians. The traffic lights have additional right arrows to independently control cars turning to the right. Furthermore, there are members of our product line with signal buttons for pedestrians to request a green phase for them. Two alternative functionalities are specified for these signal buttons:

- an immediate switching to green, and
- a switching after a specific time period to ensure that cars will get a minimum time span for their green phase.

The corresponding variability is depicted in the feature model in Figure 3.

![Feature model of a traffic lights product line](Image)

Figure 3. Feature model of a traffic lights product line.

Based on the feature model and its variability, a logical architecture (often called a function net) is specified in a next development step (Figure 4). It is a structural layout of the product line in the form of functions with input and output ports, communication relationships, and function variants [12]. The output of this modeling process is the starting point for the detailed specification of the behavior with the help of MATLAB/Simulink models.

Our prototypical tool v.control.mbd is capable of generating a variant-rich Simulink model out of function nets created for example in v.control [12], [13], [16], which is another prototype implemented in previous research activities. The result of this translation process is a Simulink model which provides the developer not only with a structural framework of the specified functions, but also with a mechanism to switch between different product line members within the Simulink model for simulation, validation or code generation purposes (see for example the translation of the XOR function ahead_signaler in Figure 4 into the corresponding Simulink specification depicted in Figure 6).

The graphical user interface (GUI) which is part of the v.control.mbd prototype (see Figure 5) allows the developer to interactively configure the Simulink model and to switch between different specified configurations. The first column
in Figure 5 shows a representation of the Simulink model structure. The further columns in the GUI represent different configurations. A row represents a subsystem (a function) in the model together with its corresponding configuration status. The configuration status can interactively be changed simply by clicking on the respective icon. The status of the corresponding subsystem in the Simulink model is automatically synchronized (see Figure 6). Thus, the configuration information is visible for the user within the Simulink model itself.

In order to simulate a particular configuration in Simulink, the user simply has to activate this configuration (e.g. the highlighted configuration immediately in Figure 5) and simulate the model.

If a configuration is changed by clicking on a status in the configuration GUI, v.control.mbd adjusts the selection status of all directly or indirectly affected systems. This way it is ensured that variability constraints resulting from the architectural design or from the user input are continuously valid. All of these features (configuration, consistency check, constraints editing, simulation) can be done in MATLAB / Simulink using the v.control.mbd plugin.

IV. RELATED WORK

The main focus of our activities is the application of product line principles together with a step-wise, function-oriented, and architecture-centric development. Such a modular and compositional approach is necessary to tackle the complexity found in the automotive domain as described in [17], for instance. AUTOSAR [1] becomes a standard in this domain: Besides the standardization of basic software in vehicles, it also consists of architecture description languages (ADLs) for the specification of application software and hardware. But it does not prescribe the development methods nor how to support variability management and product line engineering. There are other ADLs and methods which can be used for development steps before AUTOSAR, e.g. [15], [8], [19]. The latter also aims to directly support the development according to the AUTOSAR philosophy.

There are already several surveys of the usage of MATLAB / Simulink for product line engineering. In [5] it was investigated how code generators interpret Simulink constructs which were used for modeling variable parts. The construction of a product model on the basis of model libraries and templates by selecting features was investigated in [11]. Which Simulink constructs are suitable to express variability, how they can be configured, and how product models can be derived from such models is investigated in [3], [7]. The approach we presented here is very similar to this work. However, we have chosen to realize a deep integration of the variability handling mechanisms within Simulink which allows for a stand-alone usage of the new functionality in Simulink as well as for the integration of Simulink within an architecture-centric
Development elements in application models are annotated with features in [6]. The features are used to specify presence conditions for the elements. The approach how application models (e.g., UML class diagrams, Simulink models etc.) are configured with the help of features is similar to our work. Also, we use an analog approach for the verification of feature models. But in contrast to [6], we have explicitly integrated the notion of variability into our application models [12]. Thus, we are able to configure and analyze the variability in application models without an external feature model.

In [18] a catalog of verification properties that are essential to a type safe composition of modules are introduced on the basis of a formal interpretation of feature models. However, they assume that each feature is implemented by a distinct module, which is not the case in our approach.

A similar approach for the configuration of variant-rich (architectural) models with the help of a feature model is implemented in [4]. As in [12], they integrate a feature model with the other models by building an internal, unified feature model which is used for an interactive configuration with feedbacks.

V. CONCLUSION

Development tasks in the automotive industry are usually very complex. Not only the complexity, but also the huge size of data to be handled during a complete product development lead to the necessity to introduce new development methods that tackle these problems. One of the most effective way to overcome the described complexity and size problem is the proactive, planned and optimized reuse of development artifacts. A high grade of reuse can be reached using the development paradigm of product lines. A necessary prerequisite to implement reuse within the development artifacts is the possibility to express and analyze variability already during the development. The approach presented in this paper allows for such an integrated and early specification and analysis of variability without adding a new paradigm or tool. Based on the general ideas of feature modeling it is possible to marry stand-alone applications extended to handle variability with the overall development process as shown in the paper. Although we present only one possible witness for such an integration, we think that it could be extended to many engineering tools in general. This would lead to a pervasive variability management approach that will find acceptance within the industrial development departments.

REFERENCES


[20] “Feature modeling plugin (version 0.7.0),” CASE-Tool (Eclipse-Plugin), University of Waterloo, Canada, 2006, see http://gsl.uwaterloo.ca/projects/fmp-plugin/.