Engineering and Assessment of Variant-rich Embedded Software*

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

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

January 12, 2010

Abstract

Product variety has steadily increased for different reasons in many industries in the past. The challenge is to manage the high degree of necessary variety while keeping the engineering and production efforts and costs under control. The answer to satisfy the need of "mass customization" has been a product line approach where a set of similar products (a product family) is based on a common platform and common assets, but still possess specific features and functionality in order to meet particular customer requirements. The solution strategy is to exploit commonality between products and efficiently handle the product variety while keeping the product distinct.

These ideas were successfully applied within the software development area. However, the application of this methodology within the automotive domain of embedded software is still not addressed completely. Although there is a need for a pervasive and consistent variability management, its introduction within an often long-established development process where users have to change their engineering praxis is challenging. We need to introduce a development process that is on the one hand flexible enough to meet the requirements of the engineers and on the other hand powerful enough to pervasively specify and analyze variability within requirements, functions, software, and hardware descriptions.

In this paper, we will describe a concrete approach for the development of automotive E/E systems which is based on the integration of product line concepts with an architecture-centric development. The main focus is set on the pervasive handling of variability and its analysis concerning concurring solutions strategies. In addition to the results which we already introduced in the Embedded World Conference 2009, the focus of the research project EBASO is expanded from functional requirements to software architectural descriptions and implementation aspects.

1 Introduction

In current automotive development projects it turns out that the product line approach gets more and more awareness not only in the software development but also in typical mechanical engineering disciplines. The benefits of using a product line approach are clearly defined in literature and are widely accepted. Nevertheless, until now there are no applications that exploit the variability knowledge during a complete development process; from requirements engineering

*This work was partially funded by the Federal Ministry of Education and Research of Germany (BMBF) in the framework of the EBASO project (German acronym for "engineering and assessment of variant rich embedded software") under grant: 01IS09022. The responsibility for this article lies with the authors. For further information cf. the project's website: http://www.prostep.com/ebaso/

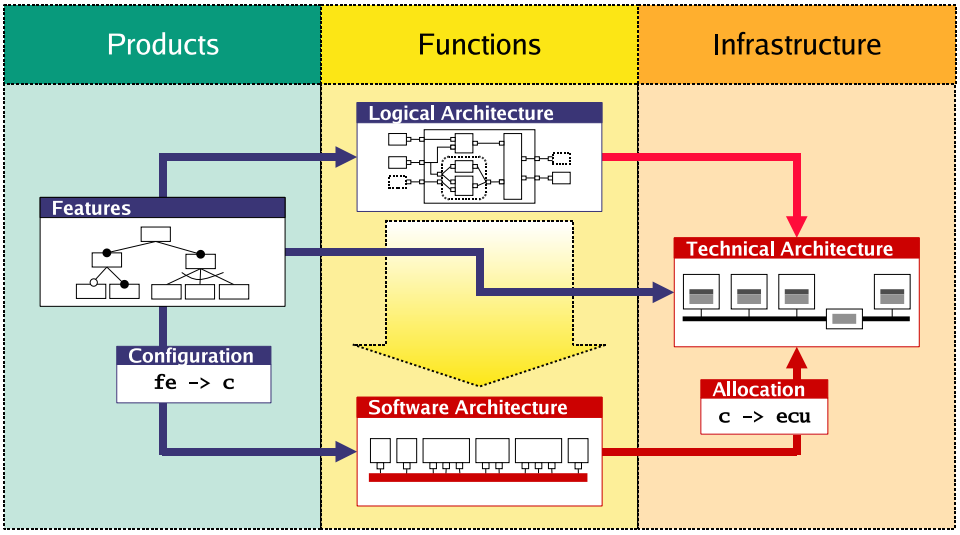


Figure 1: Artifacts of the VEIA reference process.

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, 24] or the *Feature Modeling Plugin* [32] demonstrate. But the *complete integration* of these techniques within the typical domain-specific development tools and the seamless transfer of variant information between different engineering tools is still not industrial practice. In the BMBF funded research project VEIA[1] a reference process was defined [14, 15] that allows for such an integrated approach as demanded above. The views in the functional swimlane (see Figure 1) are different abstractions of a functional view onto systems. They are related by realization or refinement relationships. Between the functional views and the infrastructural views there is an allocation mapping or deployment relationship. Thus, function nets are used for the specification of the functionality which has to be implemented within the underlying layers as there are a software and a hardware layer. A feature model is used for the abstract specification of the wanted variability which should be implemented within the product line. It can be used to control the variability within the other development artifacts.

Within the research project EBASO this approach is extended with respect to an exemplified integration of variability within the specialized engineering tool *MATLAB/Simulink* [31]. Thus, developers have the possibility to use this tool as a stand-alone engineering tool for the development of their applications within the already known framework. The advantage comes with the integrated variability management that was completely adapted to the usual development practice in MATLAB/Simulink. 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. Thus, the user is not forced to be trained on a new tool. The usage of the new functionality is intuitive and it was realized as close as possible to the usual development steps within Simulink as shown in section 3.

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

2 Model-based variability management

Variability management needs to be done continuously during the development process: from requirements to solutions in form of software and hardware. In the beginning, it is necessary

[1] VEIA is the German acronym for “distributed development and integration of automotive product lines”.

to define what variability needs to be supported by the product line under development, what products should be developed within the product line, what they differ in, and what they have in common. Then, a step-wise realization follows, where on different abstraction levels the functionality is determined, and the variability is analyzed to commit to the most appropriate solution under the given circumstances. And finally, the product line needs to be implemented by both software and hardware.

AUTOSAR [1] becomes a standard for the development of automotive software systems: Besides the important standardization of basic software in vehicles, it also provides the means to describe the application software, the technical architecture and the allocation of software onto hardware (cf. Figure 1).

An ideal development process begins with the collection of requirements, continues with the commitment on functions to be implemented (described and organized within a function net), and finally, the realization is designed by specifying software components which will be installed on and executed by electronic control units (ECUs). Variability occurs in all these development artifacts. Therefore, we need means to manage it continuously.

Feature models [7, 16, 17, 19] are appropriate to capture variability requirements or solutions on a high abstraction level. They are widely used and already supported by tools, e.g. *pure::variants*, *Feature Modeling Plugin*.

To capture variation in functions, data and control flow we suggest to use functional architecture models which are capable of (formally) analyzing the variability to guide the realization process. The prototype *v.control* [20, 21, 26] supports the management of function nets for product lines: A function net consists of functional components that may be mandatory or optional. Furthermore, there are XOR functions which encapsulate functional alternatives. Functions may also vary with respect to their interface: there may be optional ports or varying signal sets on ports. This variability may be controlled with a connected feature model. Hereby it is possible to control the variability of the functionality by selecting (or deselecting) features in the feature model. This step is often called configuration (of the variability). A valid and complete configuration represents a specific product of the product line.

In *v.control* it is possible to represent feature models, function nets and their interrelation. Furthermore, these models can be jointly configured and analyzed [20]. The application of metrics for assessment purposes is also supported, as already described in [21].

A next step is to detail the specification and to prepare the implementation steps. This can be done by behavioral specifications which can be simulated or used to generate software code. For this, we use the commercially available tool suite MATLAB / Simulink. With the help of the *v.control.mbd* plugin we are able to represent and configure the above mentioned variant-rich function nets as Simulink models. This allows the developer to simulate the models as well as to use available code generators to produce software code.

3 Variability management in MATLAB / Simulink

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, 10]. 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

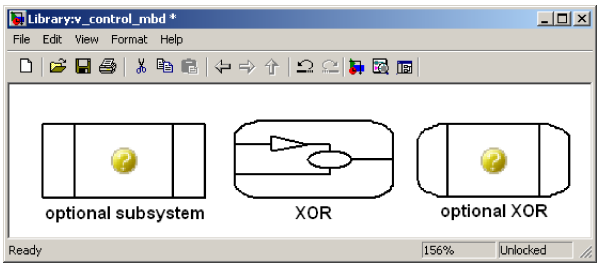


Figure 2: Elements of the *v.control.mbd* block library in Simulink.

- 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 2): 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. Finally there 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 5b). 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.

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 3). 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.

The third part of the *v.control.mbd* plugin is the configuration and analysis user interface shown in Figure 5a. 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.[2] The result of one or more configuration steps is a configuration consisting of a selection status for every subsystem. We distinguish between the following selection statuses (see enumeration type `Selection` in Figure 3):

- `selected_explicit`: The user has explicitly selected this subsystem (green check mark[3])
- `deselected_explicit`: The user has explicitly deselected this subsystem (red cross[3])
- `selected_derived`: a constraint-based deduced selection (gray check mark[3])
- `deselected_derived`: a constraint-based deduced deselection (gray cross[3])

[2] A binary decision diagram (BDD) based engine is used in the current implementation.
[3] in Figure 5

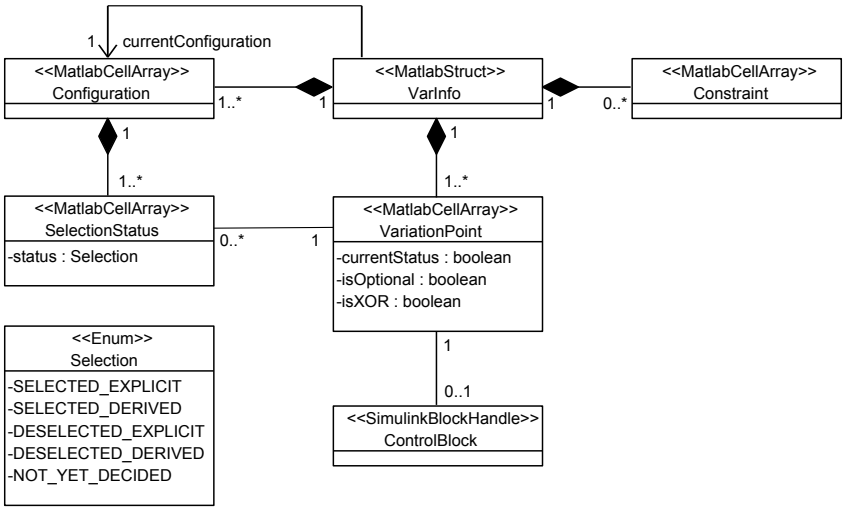


Figure 3: Data model of *v.control.mbd* in Simulink.

- `not_yet_decided`: No decision has been made yet (yellow question mark)

The user is guided during the configuration process to guarantee that the resulting configuration is consistent and respects all the automatically generated constraints. Additional constraints can as well be added by the user in the GUI.

As sketched in section 2, the usage of *v.control.mbd* is embedded in our reference process. Like the function nets of our logical architecture (Figure 1), a Simulink model also consists of hierarchical subsystems with input and output ports, connections between the ports (called *lines* in Simulink) and behavioral specifications. This allows us to generate Simulink models from function nets.

The *v.control.mbd* implementation is capable to generate a Simulink model based on a function net which was modeled in the *v.control* prototype. Currently, the generated models do not contain any system behavior, but they are a starting point for the behavioral modeling in Simulink. However, the variability modeled in the function net is transferred to the Simulink model and is managed by the *v.control.mbd* plugin. The following section presents a small example of the usage of *v.control.mbd*.

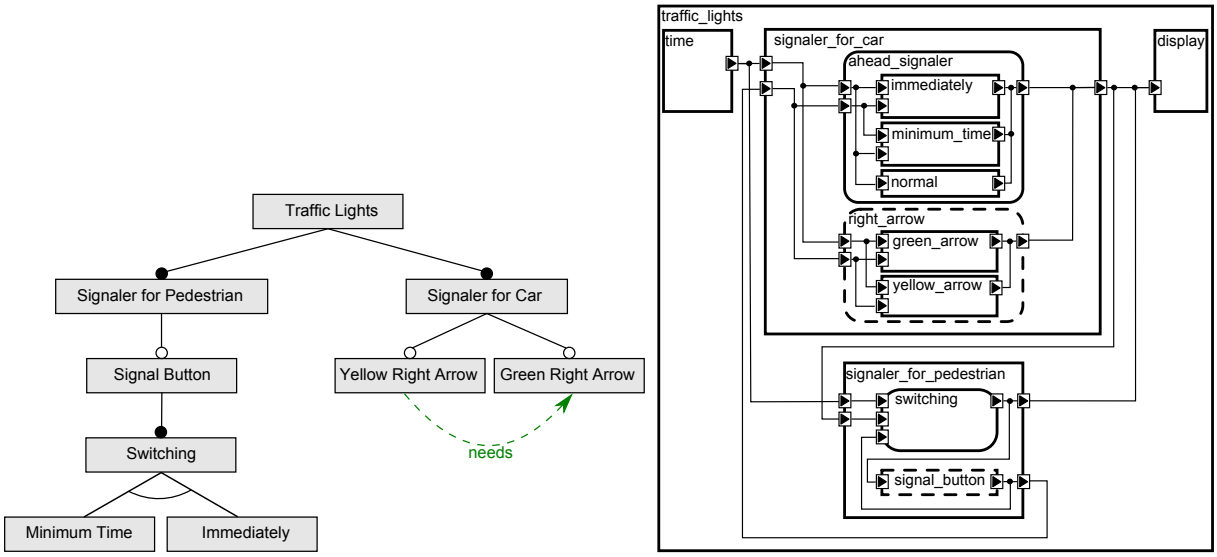
4 Example

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. Also, some of the members of our product line have signal buttons for pedestrians to request a green phase for them. We specify two alternative sequences how the pedestrian light switches to green:

- 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 4a.

Based on the feature model, a logical architecture (a function net) is defined as shown in Figure 4b. Within this development step a structural layout of the product line, its communication relationships and its variants are defined which gives us a starting point for the detailed specification of the behavior with the help of MATLAB / Simulink.



(a) Feature model.

(b) Function net.

Figure 4: Specification of a traffic lights product line.

Our prototypical tool *v.control.mbd* is capable of generating a variant-rich Simulink model based on function nets defined in *v.control*. The resulting Simulink model consists of a structural framework of the specified functions. For example, see how the XOR function `ahead_signaler` in Figure 4b is translated into the corresponding Simulink specification depicted in Figure 5b.

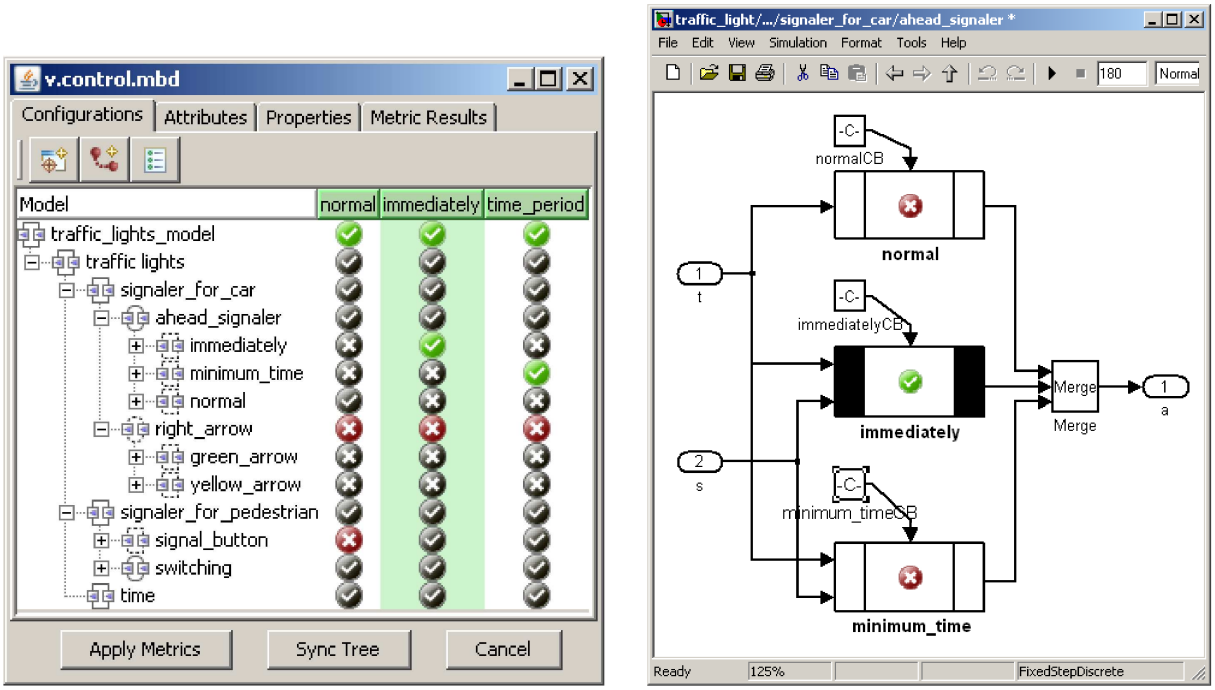
The graphical user interface (GUI) allows the developer to interactively configure the Simulink model and to switch between different specified configurations. The first column in Figure 5a shows a representation of the Simulink model structure. The other columns in the GUI represent different product configurations. Each row represents a subsystem in the model (a function) and its corresponding configuration status. The configuration status can be changed simply by clicking on the icon. This will automatically change the status of the corresponding subsystem in the Simulink model and will enable or disable this element as depicted in Figure 5b. Thus, the configuration information is visualized within the Simulink model itself.

The user has the possibility to switch between different configurations and to activate exactly one configuration which may be simulated in Simulink afterwards. As you can see in Figure 5a, the active configuration `immediately` is highlighted (by a green shaded background). Because the subsystem `immediately` has been explicitly selected by the user in this configuration, the signalers for cars will immediately switch to red when a pedestrian pushes the signal button.

When a configuration is changed by clicking on a status in the configuration GUI, *v.control.mbd* adjusts the selection status of all subsystems ensuring that constraints and rules derived from the architectural model are continuously valid. In addition, the user can define additional constraints directly in the GUI and check the new specification for consistency. All of these features (i.e. configuration, consistency check, constraints editing, simulation of the active configuration) can be done directly in MATLAB / Simulink with the help of the *v.control.mbd* plugin.

5 Related work

The main focus of our activities is the application of product line principles together with a stepwise, 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 [27], for instance. AUTOSAR [1] becomes a standard in this domain: On the one hand, it standardizes the basic software in vehicles (the hardware abstraction layer for the application



(a) User interface for the configuration of a Simulink model.

(b) The configured Simulink model (extract): XOR variation point `ahead_signaler`.

Figure 5: Extensions of MATLAB / Simulink for variability support.

software), on the other hand it also provides the description means to specify automotive systems in terms of both application software and hardware architectures. The main purpose of the description language is the exchange of specifications between development partners. However, the AUTOSAR standardization does not prescribe any specific development processes and methods. The current status of AUTOSAR also does not contain a direct support of variability management issues besides a low level calibration and configuration. Our emphasis is to provide methods and techniques for a product line engineering support which are compatible to AUTOSAR.

Architecture description languages (ADLs) are widely used to specify systems and software designs in the large. There exist quite a lot of approaches, e.g. see [22]. However, only a few support embedded or automotive software engineering on different abstraction levels, e.g. [5, 12, 25, 28, 30], or variability management, e.g. [11, 23, 30]. There is a tool support for some of these approaches, but they are mostly isolated solutions which are not tightly connected to standard development tools found in the automotive industry.

There are already several surveys of the usage of MATLAB / Simulink for product line engineering. In [6] 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 [18]. 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, 10]. 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 methodology [14].

Model elements in application models are annotated with features in [8]. 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 [8], we have explicitly integrated the notion of variability into our application models [20]. Thus, we are able to configure and analyze the variability in application models without an external feature model.

[13] also introduces an approach of "implementing" variability on model level. They distinguish between positive and negative variability which they can also control with the help of features from feature models. The former kind of variability is realized by using aspect-oriented techniques in which additional parts are weaved into the common model platform. The latter is realized by removing optional parts from a given model base. According to their terminology, the approaches in [8] and [4] as well as our approach are examples of implementing negative variability on model level in which features are connected to model elements. The selection (or deselection) of a feature affects the occurrence of the linked model element in the resulting product model. As in [8], variability is not an explicit and distinctive modeling concept in their application models. The tool support or the process has to ensure that a valid configuration of a feature model will result into a well-formed and valid product model on application model level. In our approach, variability is explicitly represented in our application models. This gives us the possibility to interpret the variability internally as a feature model. By this, the verification if a configuration is satisfiable is based on a pure check of feature model configurations.

In [29] 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 [20], 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.

6 Ongoing and future work

The approach presented here is work in progress. The introduced adaptation of MATLAB / Simulink is a first step. We aim to integrate and implement further analyzing and assessment methods into our approach. Especially the use of metrics, as already applied on the function net level [21, 26], is one of the main goals in EBASO. The definition and implementation of an application interface for the assessment of variant-rich Simulink models is under progress. Additionally, the selection, definition, and finally the implementation of useful metrics is future work.

An interesting and important aspect in product line engineering is the consideration of variability binding times. The binding time is the latest moment in the development life-cycle at which engineers can bind a variation point [9]. In general, realization mechanisms are highly interrelated to the binding time of variability. An interesting, but first sketch of the specification of binding times in Simulink is given in [3]. They analyzed the relation to code generation options provided by a specific, commercially available code generator (namely *dSpace TargetLink*). Up to now, the explicit consideration of binding times was out of focus. But it is a necessary task for the assessment of alternative solutions for the realization of a product line.

7 Conclusion

Development tasks in 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 therefor 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. Based on the general ideas of feature modeling, it is possible to combine stand-alone applications with an overall development process to handle variability as we have shown for the development of embedded software with Simulink in this paper. Although we have presented 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

[1] AUTOSAR Development Partnership: *AUTOSAR – AUTomotive Open System ARchitecture.* `http://www.autosar.org/`.

[2] D. Beuche: *Modeling and building software product lines with pure::variants*. In *Proc. 12th Int. Software Product Line Conf. (SPLC 2008)*, pp. 358–358, Limerick, Ireland, Sept. 2008. IEEE Computer Society, ISBN 978-0-7695-3303-2.

[3] D. Beuche and J. Weiland: *Managing flexibility: Modeling binding-times in Simulink*. In R.F. Paige, A. Hartman, and A. Rensink (eds.): *Proc. 5th European Conf. on Model Driven Architecture – Foundations and Applications (ECMDA-FA 2009)*, pp. 289–300, Enschede, The Netherlands, June 2009. Springer-Verlag, ISBN 978-3-642-02673-7. LNCS 5562.

[4] G. Botterweck, A. Polzer, and S. Kowalewski: *Interactive configuration of embedded systems product lines*. In *Proc. Int. Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009)*, San Francisco, California, USA, Aug. 2009. `http://www.lero.ie/maple2009/`, colocated with the 13th Int. Software Product Line Conf. (SPLC 2009).

[5] P. Braun, M. von der Beeck, U. Freund, and M. Rappl: *Architecture centric modeling of automotive control software*. In *Proc. SAE 2003 World Congress of Automotive Engineers*, SAE Transactions 2003-01-0856, Detroit, USA, Mar. 2003. `http://www.forsoft.de/automotive`.

[6] S. Bunzel, U. Judaschke, and E. Kalix: *Variant mechanisms in model-based design and code generation*. In *Proc. MathWorks Int. Automotive Conf. (IAC 2005)*, 2005.

[7] K. Czarnecki and U.W. Eisenecker: *Generative Programming – Methods, Tools and Applications*. Pearson Education. Addison-Wesley, 2000.

[8] K. Czarnecki and K. Pietroszek: *Verifying feature-based model templates against well-formedness OCL constraints*. In *Proc. 5th Int. Conf. on Generative Programming and Component Engineering (GPCE 2006)*, pp. 211–220, Portland, Oregon, USA, Oct. 2006. ACM, ISBN 1-59593-237-2.

[9] S. Deelstra, M. Sinnema, and J. Bosch: *Variability assessment in software product families*. Elsevier Journal on Information and Software Technology, 51(1):195–218, 2009.

[10] C. Dziobek, J. Loew, W. Przystas, and J. Weiland: *Von Vielfalt und Variabilität – Handhabung von Funktionsvarianten in Simulink-Modellen*. Elektronik automotive, pp. 33–37, Feb. 2008. (title in English: "Model diversity and variability – handling of functional variants in Simulink models").

[11] P. Feiler: *Modeling of system families*. Tech. Note CMU/SEI-2007-TN-047, CMU-SEI, July 2007. `www.aadl.info`.

[12] P.H. Feiler, D.P. Gluch, and J.J. Hudak: *The architecture analysis & design language (AADL): An introduction*. Tech. Rep. CMU/SEI-2006-TN-011, CMU-SEI, Feb. 2006. `www.aadl.info`.

[13] I. Groher and M. Voelter: *Expressing feature-based variability in structural models*. In *2nd Workshop on Managing Variability for Software Product Lines: Working With Variation Mechanisms*, Kyoto, Sept. 2007. In conjunction with the 11th Int. Software Product Line Conf. (SPLC 2007).

[14] M. Große-Rhode: *Architecture-centric variants management for embedded systems. Results of the project 'Distributed Development and Integration of Automotive Product Lines'*. ISST-Report 89/08, Fraunhofer ISST Berlin, Oct. 2008. `http://veia.isst.fraunhofer.de/`.

[15] M. Große-Rhode, S. Euringer, E. Kleinod, and S. Mann: *Rough draft of VEIA reference process*. ISST-Report 80/07, Fraunhofer ISST Berlin, Jan. 2007. `http://veia.isst.fraunhofer.de/`.

[16] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson: *Feature-oriented domain analysis (FODA) – feasibility study*. Tech. Rep. CMU/SEI-90-TR-21, ADA235785, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1990.

[17] K.C. Kang, J. Lee, and P. Donohoe: *Feature-oriented product line engineering*. IEEE Software, 19(4):58–65, 7/8 2002.
[18] S. Kubica: *Variantenmanagement modellbasierter Funktionssoftware mit Software-Produktlinien*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Informatik, 2007. Bd. 40, Nr. 4.
[19] K. Lee, K.C. Kang, and J. Lee: *Concepts and guidelines of feature modeling for product line software engineering*. In C. Gacek (ed.): *Software Reuse: Methods, Techniques, and Tools: Proc. of the 7th Int. Conf. on Software Reuse (ICSR 7)*, vol. 2319 of *LNCS*, pp. 62–77. Springer-Verlag, 2002. Austin, USA, April 15-19, 2002.
[20] S. Mann and G. Rock: *Dealing with variability in architecture descriptions to support automotive product lines*. In D. Benavides, A. Metzger, and U. Eisenecker (eds.): *Proc. 3rd Int. Workshop on Variability Modeling of Software-intensive Systems (VAMOS 2009)*, pp. 111–120, Sevilla, Spain, Jan. 2009. ICB-Research Report No. 29, ISSN 1860-2770 (Print); 1866-5101 (Online).
[21] S. Mann and G. Rock: *Dealing with variability in architecture descriptions to support automotive product lines: Specification and analysis methods*. In *Proc. embedded world Conference 2009*, Nürnberg, Germany, Mar. 2009. WEKA Fachmedien, ISBN 978-3-7723-3798-7. ISBN 978-3-7723-3798-7.
[22] N. Medvidovic and R.N. Taylor: *A classification and comparison framework for software architecture description languages*. IEEE Transactions on Software-Engineering, 26(1):70–93, Jan. 2000.
[23] R.C.v. Ommering, F.v.d. Linden, J. Kramer, and J. Magee: *The Koala component model for consumer electronics software*. IEEE Computer, pp. 78–85, Mar. 2000.
[24] pure::systems GmbH: *pure::variants*. CASE-Tool. `http://www.pure-systems.com/`.
[25] M. Rappl: *Entwurfsorientierte Modellierung eingebetteter Systeme*. PhD thesis, Technische Universität München, 2004, ISBN 3-8325-0665-9. Logos-Verlag.
[26] G. Rock and S. Mann: *Assessment of product line architecture descriptions in v.control*. In I. Schieferdecker and S. Goericke (eds.): *Software Quality Engineering – Proc. of the CONQUEST*, pp. 163–178, Nürnberg, Germany, Sept. 2009. dpunkt.verlag, ISBN 978-3-89864-637-6. ISBN 978-3-89864-637-6.
[27] J. Schäuffele and T. Zurawka: *Automotive Software Engineering*. ATZ-MTZ-Fachbuch. Vieweg, 2003.
[28] SysML Partners: *Systems Modeling Language (SysML) Specification (Version 1.1)*, Nov. 2008. `www.sysml.org`, OMG Document Number: formal/2008-11-01. Standard document URL: `http://www.omg.org/spec/SysML/1.1`.
[29] S. Thaker, D. Batory, D. Kitchin, and W. Cook: *Safe composition of product lines*. In *Proc. 6th Int. Conf. on Generative Programming and Component Engineering (GPCE 2007)*, pp. 95–104, Salzburg, Austria, Oct. 2007. ACM, ISBN 978-1-59593-855-8.
[30] The ATESST Consortium: *EAST ADL 2.0 specification*. Specification 2008-02-29, 2008. `www.atesst.org`, Draft.
[31] The MathWorks: *Simulink 7*. CASE tool, 2009. a MATLAB toolbox, see `http://www.mathworks.com/`.
[32] University of Waterloo, Canada: *Feature modeling plugin (version 0.7.0)*. CASE-Tool (Eclipse-Plugin), 2006. see `http://gsd.uwaterloo.ca/projects/fmp-plugin/`.