Model-Based Analysis of Design Artefacts Applying an Annotation Concept

Daniel Merschen*, Yves Duhr**, Thomas Ringler**, Bernd Hedenetz**, Stefan Kowalewski*
*Embedded Software Laboratory, RWTH Aachen University
Aachen, Germany
{merschen | kowalewski}@embedded.rwth-aachen.de

Group Research & Advanced Engineering, Daimler AG**
Boeblingen, Germany
{yves.duhr | thomas.ringler | bernd.hedenetz}@daimler.com

Abstract: In automotive software development, dependencies among process artefacts, i. e. requirements, implementation and test cases, are often not obvious. This causes time-intensive manual analysis efforts to incorporate changes during software evolution. Therefore, automated tool support is essential to establish an efficient change management during the software life cycle.

This paper presents a model-based concept which integrates the artefacts themselves as well as development-related meta information about them to establish both functional and process-related artefact analyses. To this end, we represent them as models in the Eclipse Modeling Framework and apply model transformations to support different kinds of automated analyses.

1 Introduction

Within the automotive industry market trends like functional innovations and an increasing number of car lines lead to a dynamic life cycle of software applications, e. g. the need to incorporate new features or changes of requirements late in the development process. Hence, a special software development approach is necessary which is better suitable to handle such dynamics exploiting systematic reuse and variability concepts.

To tackle these challenges, Daimler's Electrical and Electronics division for the body and comfort domain has been following the model-based approach with Matlab/Simulink [Matb] for several years [WDR08]. As Figure 1 visualises the product line is first modelled as a feature tree following the Feature-Oriented Design Analysis (FODA) [KCH+90]. This feature tree supports the engineer in an early stage by documenting the different dependencies. Second, the artefacts, i. e. a system specification (requirements), a Simulink model (implementation model) and a test specification, are built accordingly. However, as described by [TDH11] the current approach has to be extended in order to efficiently manage the increasing complexity of future functions, especially to support software *evolution*. One intuitive step towards this evolution support is already realised by handwritten documentation which is added to the subsystems which implement special features

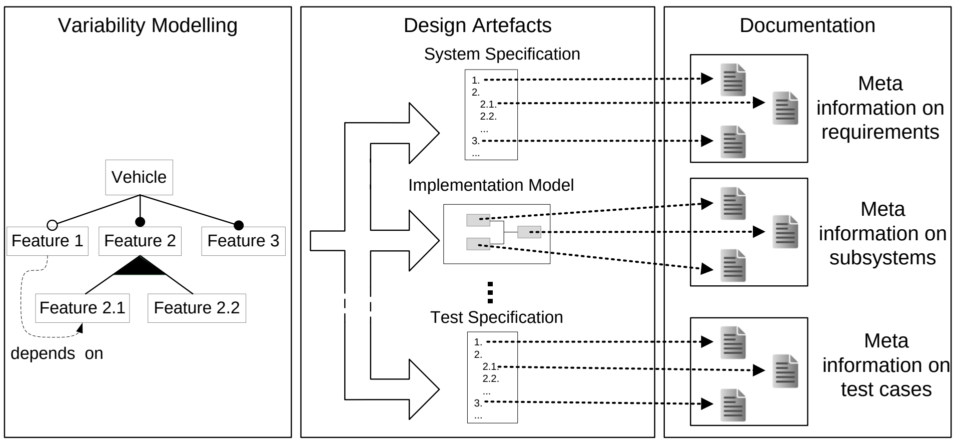
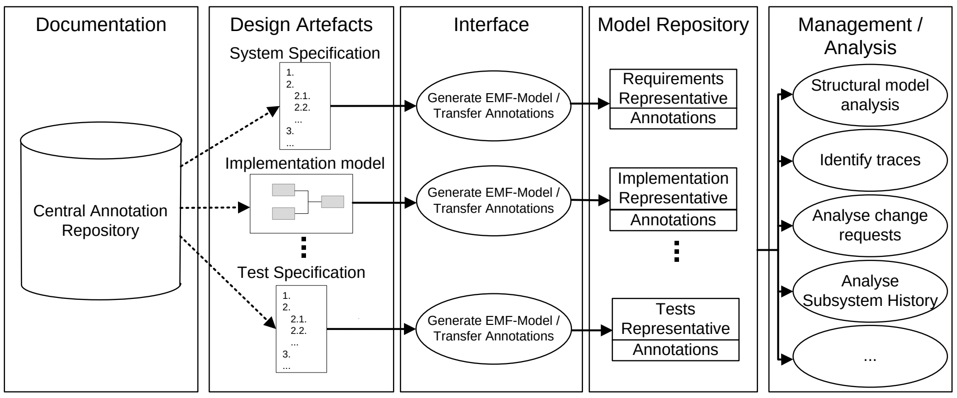


Figure 1: Workflow of the process described by Thomas et al. [TDH11]

in the Simulink model respectively the requirements or test cases (cf. right part of Figure 1). Nevertheless, due to the complexity of the product line artefacts and the documentation style (free text) maintenance and evolution are still time-consuming. This phenomenon is well-known in literature. As [Som07] points out the costs of maintenance and evolution of long-lifetime software systems exceed the development costs by factor 3-4. According to [MD08] 50% of development costs are needed just to understand legacy code.

To our mind, it is therefore important to provide engineers with automated analyses to manage software evolution. These analyses should first consider functional and structural aspects of artefacts to handle the artefact complexity, e. g. to create views on a Simulink model which visualise model elements depending on a given signal. Second, process-related and variability-related issues have to be taken into account, e. g. to identify reasons for changes of artefacts during life cycle. In [PMT^{+}10] and [MPBK11] we focus on structural and functional analyses of single and multiple artefacts, while this paper will describe a concept to integrate process-related information into the different artefacts to widen the set of possible analyses by process-related ones. To this end, we have elaborated a general artefact integration concept as well as an annotation concept to capture meta information. This concept was inspired by the model-based development of embedded on-board software at Daimler AG and is explained in Section 2. That is why we mainly focus on the applied tools (Matlab/Simulink [Matb] and IBM Rational DOORS [IC]) here. However, the concept is held abstract enough to be generalisable for the use in other tools and application areas as well which will be explained in Section 2.4. In Section 3 we discuss the benefits based on a case study of experienced Daimler engineers. We describe related work in Section 4 before summarising the paper and giving an outlook on the next steps in Section 5.



Documentation
Central Annotation Repository
Design Artefacts
System Specification
Implementation model
Test Specification
Interface
Generate EMF-Model / Transfer Annotations
Generate EMF-Model / Transfer Annotations
Generate EMF-Model / Transfer Annotations
Model Repository
Requirements Representative
Annotations
Implementation Representative
Annotations
Tests Representative
Annotations
Management / Analysis
Structural model analysis
Identify traces
Analyse change requests
Analyse Subsystem History
...

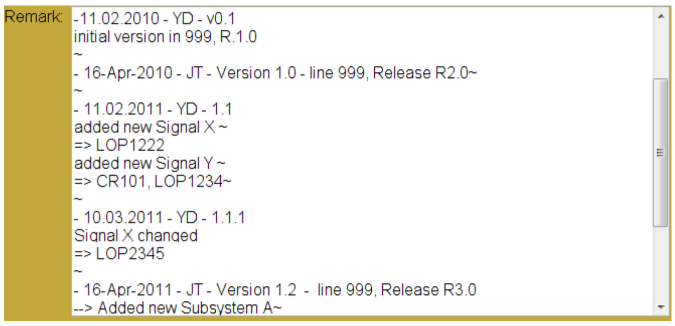
Figure 2: Conceptual overview to establish multi-artefact analyses consider also meta information

2 Approach

To tackle the challenges mentioned in Section 1 it is necessary to add automatically analysable process-related meta information to the artefacts and to connect the different tools (cf. [BFH$^+$10]). We do this by following the concept description in Figure 2. To integrate meta information we elaborated a concept based on a central annotation repository. Section 2.1 will describe how this concept works. Subsequently, we integrate the different artefacts into one model repository. This step is described by the middle three parts of Figure 2 and will be explained in detail in Section 2.2. Based on integrated artefacts including meta information different kinds of analyses are planned to be realised (right part of Figure 2) one of which is outlined in Section 2.3. Finally, we will describe how the presented concept can be generalised in Section 2.4

2.1 Documentation of Design Artefacts

In order to efficiently incorporate late changes and to support evolution we would like to be able to analyse artefacts not only with respect to their structural and functional construction but also consider meta information about the development process like changes of subsystems due to change requests, bug reports or combinations of them. In model-based development with Matlab/Simulink such information is often specified as free text within special version information blocks from TargetLink [dSp] libraries. Hence, these blocks do not implement functionality but just answer the purpose of documentation. Figure 3 exemplifies the content of a version information block. Currently, diverse meta information is specified that way within these blocks, e. g. time stamp and author of the meta information itself, the version, the information is relevant for, the ID of a change request that initiated the change and the production line(s) the subsystem is relevant for.



```
Remark: -11.02.2010 - YD - v0.1
        initial version in 999, R.1.0
        ~
        - 16-Apr-2010 - JT - Version 1.0 - line 999, Release R2.0~
        ~
        - 11.02.2011 - YD - 1.1
        added new Signal X ~
        => LOP1222
        added new Signal Y ~
        => CR101, LOP1234~
        ~
        - 10.03.2011 - YD - 1.1.1
        Signal X changed
        => LOP2345
        ~
        - 16-Apr-2011 - JT - Version 1.2 -  line 999, Release R3.0
        --> Added new Subsystem A~
```

Figure 3: A screenshot of a version information block for the documentation of subsystems

One approach to integrate this free text information into analyses would be to define a unique syntax for the free text so that a parser could collect the relevant information. In earlier work [MPBK11] we present an analysis which follows this approach. However, as, in practice, every user uses his own syntax parsing this text is error-prone. A further problem of using version information blocks is that the information can only be retrieved manually by stepping into the subsystems recursively. There is currently no way to receive an overview over subsystems and their documentation or realisation status, which complicates, for example, an estimation of the (remaining) effort to incorporate a change request. Furthermore, artefact elements that are related to the same functionality are likely to be annotated with similar text. The different syntax will complicate the identification of these relationships. Uniquely defined annotations, however, could facilitate it and hence, also support the establishment of traceability. In the following section we will define what is meant by an annotation in our context and explain which requirements an annotation concept has to meet to be useful and applicable in industrial practice. In Section 2.1.2 we will outline how we realised these requirements.

2.1.1 Requirements on the Annotation Concept

To address these challenges we suggest annotations on artefacts. To our mind, an annotation concept should (1) facilitate automated analyses, (2) ensure uniqueness of annotations, (3) allow artefact-comprehensive annotating and (4) allow for an overview over all annotations.

To facilitate automated analyses it is important to ensure that there is a unique syntax for each semantics and, hence, to limit free text specification as much as possible. This requirement cannot be covered by the current free text annotation. Hence, we provide a type for each annotation such that an annotation can be defined as a tuple <TYPE, VALUE>.

In order to ensure that annotation types are unique and artefact-comprehensive our concept provides a *central* management, i. e. there is a central repository of all available types together with a description about their meaning (cf. left part of Figure 2). Each tool should then connect to this central repository to retrieve these types such that the user can only select the desired type and then specify the desired annotation value (cf. second part of Figure 2).

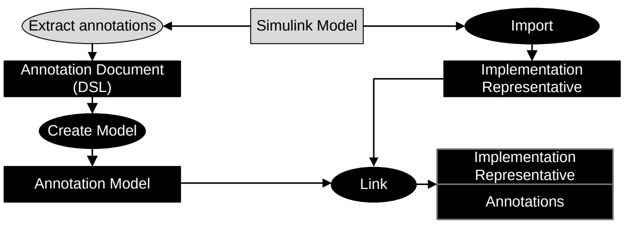

Extract annotations
Simulink Model
Import
Annotation Document (DSL)
Implementation Representative
Create Model
Implementation Representative
Annotation Model
Link
Annotations

Figure 4: Import of an annotated Simulink model into our framework

2.1.2 Realisation

Up to now, we have focussed on the annotation of Simulink models. Each block in a Simulink model contains a parameter called *UserData*, which we use to annotate it with the help of the *set_param* command of the Matlab API [Mata]. To prevent the user from arbitrarily defining annotation types himself which would limit automatising analyses the annotation is performed via a Matlab script which reads out the available types from a database (cf. left part of Figure 2) such that the user can select one. Subsequently, the user is prompted to specify the annotation value. This workflow is repeated until the user finishes annotating. Finally, the script writes the annotation to the Simulink model with the above mentioned *set_param* method.

2.2 Artefact Integration

In order to establish analyses including multiple artefacts it is necessary to connect the different tools, i. e. to join the artefacts in one common repository. With this approach we can avoid the problem of tool-dependent model analyses. The corresponding process is visualised by the middle three parts of Figure 2. We currently import Simulink models and a csv-export of the requirements in IBM Rational DOORS with Xtext [EFd], a framework to create domain-specific languages. The resulting models are models of the Eclipse Modeling Framework (EMF) [EFa], which we call *representatives* of the original artefacts and which are stored in the model repository.

Besides the artefacts themselves, the annotations, which were added to them as described in Section 2.1, have to be transferred to the corresponding representative in the model repository. Otherwise the management component described later on could not consider them for analyses. Up to now, we realised this step for Simulink models. As Matlab/Simulink does not store them in clear text within the file we follow the process in Figure 4 to transfer the annotations from Matlab/Simulink to our model repository. The grey parts describe activities which take place within Matlab/Simulink while the black ones are performed within our framework. First of all, the annotations are extracted out of the Simulink model with Matlab methods, i. e. with a Matlab script which reads all available annotations and writes them into a text file following a domain-specific language (DSL) (cf. Listing 1).

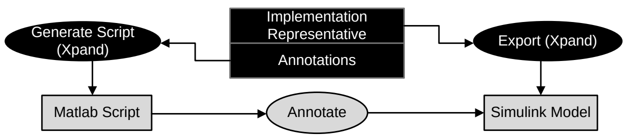


Figure 5: Export of an annotated representative implementation model of our framework to Matlab/Simulink

```
AnnotationModel AnnotModel {
  annotations {
    Annotation 0 {
      path     "ADJ_tl06/ADJ/Subsystem/Subsystem/Subsystem/ADJ_Function/SoCProcessing"
      type     "VERSION"
      value      "1.0"
      author     "JT"
      timestamp      "2010/04/15  08:09:08"
    },
    [...]
}
```

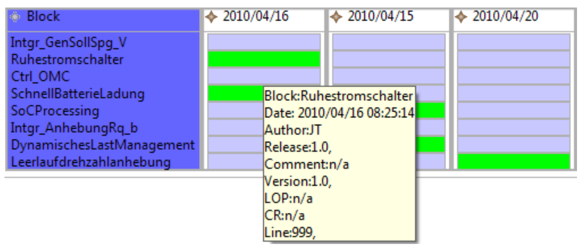
Listing 1: An excerpt of an annotation document exported via a Matlab script

For each annotation the script captures the author, the time stamp, type and value as well as the path to the annotated block. The path describes the navigation through the Simulink model via the subsystems to the block starting at the root system (analogous to a path to a file in a file system). It is necessary for the further process to find the corresponding block in the representative implementation model within the model repository of the framework.

Next, the file is parsed by an Xtext parser leading to an EMF-based annotation model, which is stored within the model repository and directly linked to the implementation representative by a model transformation (cf. Figure 2).

Results of an artefact analysis might also be stored as annotations within the representatives. For instance, one could be interested in all subsystems of a Simulink model that were affected by a change request. In this case one could append a suitable annotation to the respective subsystems of the representative. In such cases it also has to be possible to transfer such annotations from the model repository to the Simulink model. Figure 5 describes the steps to be followed for that purpose.

To this end, we apply XPand [EFc] to (1) export the implementation representative to Matlab/Simulink and (2) to generate an annotation script for it from the annotation model. Subsequently, we run the generated script to transfer the annotations to the exported Simulink model.



Block	2010/04/16	2010/04/15	2010/04/20
Intgr_GenSollSpg_V			
Ruhestromschalter	🟩		
Ctrl_OMC			
SchnellBatterieLadung	🟩		
SoCProcessing		🟩	
Intgr_AnhebungRq_b			
DynamischesLastManagement		🟩	
Leerlaufdrehzahlanhebung			🟩

```
Block:Ruhestromschalter
Date: 2010/04/16 08:25:14
Author:JT
Release:1.0,
Comment:n/a
Version:1.0,
LOP:n/a
CR:n/a
Line:999,
```

Figure 6: Resulting table of a history analysis visualised in a graphical editor of the GMP

2.3 Artefact Management and Analysis

In the right part of Figure 2 we mention some examples for analyses which base on the model repository. For details and examples about structural model analyses we refer the reader to previous work [MPBK11, PMT$^+$10]. To estimate the potential of the presented annotation concept for automated, process-related model analysis we focus here on the analysis of the subsystem history which generates a tabular overview over subsystems of a Simulink model and the dates of their changes in a graphical editor of the Graphical Modelling Project (GMP) [EFb]. Furthermore, for each date of change detailed information about the change is collected and displayed to the user (cf. Figure 6). Previously, we retrieved this information from the version information blocks mentioned in Section 2.1 by parsing the free text with an Xtext parser. However, when we applied the concept on real productive models this approach turned out not to be scalable enough for industrial practice due to too different kinds of syntax used for the free text. Now we have applied the described annotation concept, i. e. we added the information of the version information blocks with annotations on the affected subsystems. As expected, the model transformation now works well such that we can assume the concept to be a benefit for automated analysis and even for linking artefacts based on annotations.

2.4 Generalisation

Up to now, we focussed primarily on the integration of annotations into the implementation model exemplified by Matlab/Simulink. However, this approach is generally not restricted to Matlab/Simulink although some modification will be necessary to transfer the concept on other tools as well.

With respect to the implementation model we assume that the basic functionality of other tools is comparable to Matlab/Simulink and that models can be transformed into Simulink models with equivalent semantics. This assumption, of course, would have to be evaluated thoroughly before the approach is transformed to a different tool platform.

Concerning requirements and test cases the situation is different. As we can only import csv files into the model repository the syntax is fixed while the semantics is not. Hence, the import will not have to be adapted if the applied tool can export the requirements as a csv file. However, as the semantics of the resulting requirements respectively test model is unknown, the model transformations have to be adapted. This step is supposed to be time-intensive.

Moreover, we have to consider how the central repository can be accessed by the applied tools. Up to now, we have established the repository access in Matlab/Simulink. However, as the repository is realised by a relational database the available annotation types can be retrieved via usual SQL queries. Hence, the concept can be integrated into each tool which is able to communicate with a relational database.

Due to the abstractness of the concept it also becomes applicable to other application areas, e. g. to support software project planning. To this end, it has to be further extended to capture more artefacts in the model repository, e. g. whole descriptions of change request, bug reports and feature models which have to be managed by project leaders.

3 Practical Experience

In order to estimate the benefits of the presented concept for the model-based development at Daimler the concept was evaluated by three Daimler engineers with 6, 10 and 15 years of experience in software development of embedded systems (max. 10 years experience in *model-based* development). To do so, they focussed on the implementation model in Matlab/Simulink of a real-world embedded software product line. The application of the concept on other artefacts like requirements and test cases has not been evaluated yet. The evaluation criteria were (1) error-proneness , (2) benefits for automatising single artefact as well as multi-artefact analyses including meta information, (3) generalisability, (4) barriers to overcome before going active.

Error-proneness One of the targets of the presented annotation concept is to reduce the use of free text in order to prevent errors during automated model analysis. The concept can be expected to meet this requirement. The assumption is based on the analysis presented in Section 2.3 which was performed on a Simulink model with 1,000 blocks and a subsystem hierarchy depth of 8. Applying the former concept which parsed the free text of the version information blocks lead to frequent parsing errors. These are caused by the fact that many engineers work at the same Simulink model and everyone uses his own syntax to specify meta information. The same model was now successfully annotated using the presented annotation concept and analysed with an adapted model transformation which should be applicable to every implementation model representative.

Expected benefits for model analyses The concept's potential to support automated model analyses with respect to meta information was estimated as high. As the annotation syntax is unique, annotated meta information becomes automatically comparable which is regarded as an essential break-through for automatising model analyses. In order to establish multi-artefact analyses the central management of annotations is important as different tools can access the same database leading to the same annotation types in different artefacts. That way artefacts can be linked based on annotated meta information. However, up to now just annotation *types* are unique as they are stored centrally and offered to the user for selection. Annotation *values* are not yet kept centrally. This limits comparisons among annotations again and should be addressed in the next steps.

Generalisability The concept is held abstract enough to be applicable to other tool chains as well. Model analyses do not operate on the original artefacts but on their representatives in the model repository. Hence, the “only” tool-specific part of the concept is the interface which will have to be adapted to integrate other tools. However, changing the interface can be expected to be more reasonable than changing a tool as the latter would mean lots of costs for migration and teaching personal. That is why an interface change can supposed to pay off.

The extensibility for further artefacts was regarded as a further advantage of the presented concept. Integrating change requests, for instance, would also make the concept applicable to whole process management tools supporting estimating efforts.

Moreover, new annotation types can be added centrally and need not be made available to the different tools individually.

Barriers for concept introduction As almost every change of process the introduction of the presented concept cannot be established without preparatory work. All currently available free text information has to be added to the artefacts first meaning lots of hours of work. However, to the engineers’ mind, the concept will pay off due to decreasing efforts in model analysis during evolution.

For real-world scenarios annotations should be made more expressive. For instance, annotations might be related to each other or even annotate other annotations. To this end, the data modelling and model transformations should be adapted accordingly. That way, the scope of model analyses could be widened even more.

4 Related Work

In [PMT^{+}10] we elaborated structural and functional analyses to generate views on Simulink models to support the engineer with modelling tasks, e. g. to analyse parts of a Simulink model which depend on a given signal. We extended this work in [MPBK11] where we also considered analyses concerning inter-artefact relationships and made a first attempt to analyse meta information. However, as the meta information was specified using free text the concept was not sufficiently scalable onto productive models due to the amount of different syntax used to express the same semantics.

[TDH11] focuses on challenges concerning the introduction of a product line approach in the context of AUTOSAR-based [AUT] development of applications for in-vehicle systems. Especially, it is pointed out that current approaches to introduce a product line and related tool chains do not sufficiently support the product line life cycle. To do so, a seamless variant management integrating different artefacts is necessary. In this context annotations as presented in this paper can be helpful to identify relationships and, hence, to support the migration.

[BFH+10] elaborates on the general problem of tool isolation in practice. They explain which challenges evolve from tool isolation in the context of traceability and consistency of artefacts. As a cause for inconsistency they identify that current interaction among tools is not deep enough. That way possibilities of reuse are too limited. They consider a comprehensive modelling theory, an integrated architectural model and an integrated model engineering environment to be fundamental prerequisites to establish a seamless model-based development process. Our approach of artefact integration addresses the isolation problem while the annotation concept intends to make the interaction deeper.

[BDT10] elaborates on model transformations concerning safety-related embedded systems. They also perform analysis of embedded systems with model transformation. However, their objective is to automate translations from architecture description in the automotive domain into a safety analysis tool called HiP-HOPS.

The framework ToolNet [ADS02] was developed by DaimlerChrysler using MOFLON [TD] focusing on the creation, the management and consistency checks of traceability links between different tools. ToolNet assists the developer in the manual construction of traceability links. In contrast to that, our approach will support the developer by automatically creating the traceability links through the annotation information.

[ZMV+03] associates objects of a model with the corresponding objects of the model in the following design step. They use annotations only for free text documentation. This documentation could be accessed by keyword search but is not intended for automated analysis.

[SVSZ01] elaborates on the problem of losing knowledge during the development process. They developed a tool called Clockwork Enrich Tool to annotate individual blocks of a Simulink model with free text. In [MZV+03] they also introduce formalisations. However, they just support few formalised relations among artefacts. Our approach provides artefacts with more formalised knowledge to automatically analyse these relations.

5 Summary

The presented work intends to support the model-based development of an embedded software product line. Two of the main challenges in this context are to overcome the barrier of tool isolation and to allow for automated analyses of artefacts and their interrelationships considering both functional aspects and process-related meta information. To our mind, the concept which is best suitable to tackle these challenges is to integrate the different artefacts into one tool-independent model repository which offers interfaces to the current tools in use. To capture meta information about the development process on the different artefacts we presented an annotation concept. By defining a syntax for annotations and a central artefact-independent repository for annotations the concept allows for automatising artefact analyses with respect to development-related and evolution-related issues. Moreover, the concept is to a large extent constructed tool-independent as both artefacts to analyse and the related annotations are kept in an external repository.

To analyse the information specified by these annotations we can export them from Matlab/Simulink into our external framework and link them with representatives of the Simulink model. We have exemplified one analysis of meta information with the help of annotations in Section 2.3.

Up to now, we are able to import Simulink models and requirements into the external framework. Furthermore, we implemented the annotation concept for Simulink models. However, in order to establish traceability using the annotation concept we have to integrate annotations into the other artefacts as well. Last but not least the annotation concept is currently just a prototype. In order to put the concept into practice it has to be extended and restructured to make annotations more expressive. For instance, annotations are currently appended to blocks in a linear, i. e. flattened, manner. That way information about relationships among themselves cannot be captured and hence, we loose information compared to the free text approach where information can be grouped textually. This requires a deeper analysis of the kinds of information to be annotated and a different way to specify them in a user-friendly manner. After restructuring annotations a graphical user interface will be needed to support the engineers during evolution of the product line.

Acknowledgement

This work was funded, in part, by the Excellence Initiative of the German federal and state governments as well as Daimler AG. Moreover, we would like to thank Christian Dziobek, Thorsten Stecker, Uwe Spieth and the anonymous reviewers for useful inputs and constructive feedback about the presented concepts.

References

[ADS02] F. Altheide, H. Dörr, and A. Schürr. Requirements to a Framework for Sustainable Integration of System Development Tools. In *EuSEC '02*, pages 53–57, 2002.

[AUT] AUTOSAR. AUTOSAR AUTomotive Open System ARchitecture. http://www.autosar.org/.

[BDT10] M. Biehl, C. DeJiu, and M. Törngren. Integrating safety analysis into the model-based development toolchain of automotive embedded systems. In *LCTES '10*, pages 125–132, 2010.

[BFH$^+$10] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526 –545, April 2010.

[dSp] dSpace. Target Link. http://www.dspace.com/en/ltd/home/products/sw/pcgs/targetli.cfm.

[EFa] Eclipse-Foundation. EMF - Eclipse Modeling Framework. http://eclipse.org/modeling/emf/.

[EFb] Eclipse-Foundation. GMP - Graphical Modelling Project. http://www.eclipse.org/modeling/gmp/.

[EFc] Eclipse-Foundation. Xpand. http://www.eclipse.org/modeling/m2t/?project=xpand.

[EFd] Eclipse-Foundation. Xtext. http://www.eclipse.org/Xtext/.

[IC] IBM-Corporation. IBM Rational DOORS. http://www-01.ibm.com/software/awdtools/doors/.

[KCH$^+$90] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. SEI Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute, 1990.

[Mata] The MathWorks, Inc. Function Reference (MATLAB). http://www.mathworks.de/help/techdoc/ref/f16-6011.html.

[Matb] The MathWorks, Inc. Simulink – Simulation and Model-Based Design. http://www.mathworks.de/products/simulink.

[MD08] T. Mens and S. Demeyer. *Software evolution*. Springer, 2008.

[MPBK11] D. Merschen, A. Polzer, G. Botterweck, and S. Kowalewski. Experiences of Applying Model-based Analysis to Support the Development of Automotive Software Product Lines. In *VaMoS '11*, pages 141–150, 2011.

[MZV$^+$03] P. Mulholland, Z. Zdrahal, M. Valasek, P. Sainter, M. Koss, and L. Trejtnar. Supporting the sharing and reuse of modelling and simulation design knowledge. In *ICE 2003*, 2003.

[PMT$^+$10] A. Polzer, D. Merschen, J. Thomas, B. Hedenetz, G. Botterweck, and S. Kowalewski. View-Supported Rollout and Evolution of Model-Based ECU Applications. In *MoMPES '10*, pages 37–44, 2010.

[Som07] I. Sommerville. *Software engineering*. International computer science series. Addison-Wesley, 2007.

[SVSZ01] P. Steinbauer, M. Valasek, Z. Sika, and Z. Zdrahal. Knowledge supported design and reuse of simulation models. In *MATLAB 2001*, pages 399–406, 2001.

[TD] TU-Darmstadt. MOFLON. http://www.moflon.org/.

[TDH11] J. Thomas, C. Dziobek, and B. Hedenetz. Variability management in the AUTOSAR-based development of applications for in-vehicle systems. In *VaMoS '11*, pages 137–140, 2011.

[WDR08] F. Wohlgemuth, C. Dziobek, and T. Ringler. Erfahrungen bei der Einführung der modellbasierten AUTOSAR-Funktionsentwicklung. In *MBEFF '08*, pages 1 – 15, 2008.

[ZMV$^+$03] Z. Zdrahal, P. Mulholland, M. Valasek, P. Sainter, M. Koss, and L. Trejtnar. A Toolkit and Methodology to Support the Collaborative Development and Reuse of Engineering Models. In *DEXA 2003*, pages 856–865, 2003.