ASIM-Workshop STS/GMMS 2014

Treffen der ASIM/GI-Fachgruppen:
Simulation technischer Systeme
Grundlagen und Methoden in Modellbildung und Simulation

20. bis 21. Februar 2014 in Reutlingen-Rommelsbach

Tagungsband

Jürgen Scheible (Hrsg.)
Ingrid Bausch-Gall (Hrsg.)
Christina Deatcu (Hrsg.)

Arbeitsgemeinschaft Simulation ASIM in der Gesellschaft für Informatik GI

ISBN 978-3-901608-42-1 ARGESIM Report 42 * ASIM Mitteilung AM 149

ARGESIM Reports

Published by **ARGESIM** and **ASIM**, Arbeitsgemeinschaft Simulation,
Fachausschuss 4.5 der GI

Series Editor:
Felix Breitenecker (ARGESIM / ASIM)
Div. Simulation, Vienna University of Technology
Wiedner Hauptstrasse 8 - 10, A - 1040 Vienna
Tel: +43-1-58801-10115, Fax: +43-1-58801-10199
Email: Felix.Breitenecker@tuwien.ac.at

ARGESIM Report 42
ASIM Mitteilung AM 149

Titel: ASIM-Workshop STS/GMMS 2014
Treffen der ASIM/GI-Fachgruppen:
Simulation technischer Systeme
Grundlagen und Methoden in Modellbildung und Simulation

Herausgeber: Jürgen Scheible
Ingrid Bausch-Gall
Christina Deatcu
Email: christina.deatcu@hs-wismar.de

ISBN 978-3-901608-42-1

ARGE Simulation News (ARGESIM)
c/o F. Breitenecker, Div. Simulation, Vienna Univ. of Technology
Wiedner Hauptstrasse 8-10, A-1040 Vienna, Austria
Tel.: +43-1-58801-10115, Fax: +43-1-58801-10199
Email: info@argesim.org; WWW: http://www.argesim.org

Druck:
WENZEL GmbH druck . kopie . media
München

Advancing Virtual Commissioning with Variant Handling

Johannes Möck[1], Jens Weiland[1]
[1]Hochschule Reutlingen, School of Engineering
Alteburgstr. 150
72762 Reutlingen, Germany
{johannes.moeck, jens.weiland}@reutlingen-university.de

Abstract: Nowadays the software development plays an important role in the entire value chain in production machine and plant engineering. An important component for rapid development of high quality software is the virtual commissioning. The real machine is described on the basis of simulation models. Therefore, the control software can be verified at an early stage using the simulation models. Since production machines are produced highly individual or in very small series, the challenge of virtual commissioning is to reduce the effort to the development of simulation models. Therefore, a systematic reuse of the simulation models and the control software for different variants of a machine is essential for an economic use. This necessarily requires a consideration of the variability which may occur between the production machines. This paper analyzes the question of how to systematically deal with the software-related variability in the context of virtual commissioning. For this purpose, first the characteristics of the virtual commissioning and variability handling are considered. Subsequently, the requirements to a so-called variant infrastructure for virtual commissioning are analyzed and possible solutions are discussed.

1 Introduction

Nowadays the software development plays an important role in the entire value chain in production machine and plant engineering. The low-cost and rapid development of high quality software has become a crucial success factor. The machine control software (NC, PLC, HMI), which has to be solved computationally, can no longer be seen as only an appendage of a machine. Special software engineering methods are necessary to master the complexity of the control software.

The virtual commissioning is an important step to shorten developmental times (time-to-market) and to improve the quality of the control software. In virtual commissioning the real machine is described based on virtual simulation models, including kinematic and behavioral models. Using this technique the control software can be verified on an early stage by means of the simulation models. In addition, simulation results can flow back into the machine design. Thus, virtual commissioning allows an early validation and optimization of the control software and the entire machine behavior. This leads to a significant reduction of commissioning time and to a considerably higher quality of the machine. In many places, the virtual commissioning is at the threshold to productive use.

Despite the great progress that has been made in recent years in the field of virtual commissioning, there are still aspects which counteract their economic productive use. An essential aspect is the effort of developing simulation models: Production machines and plants are often highly individually and produced in very small batches. As a result, the functionality of a machine has to be customized for the respective customer. In addition, from a technical point of view, the machine-specific sensors and actuators lead to an individual control of the machine. Functional and technical differences between the different machine variants are reflected inevitably in the control software and the simulation models of each machine variant.

Depending on the developmental phase and the scope of use, the developmental effort can be reduced if parts of the control software and the simulation models could be reused for different variants of the machine. However, a systematic reuse requires consideration of the variability that can occur between the production machines. As it has been used in mechanical and plant engineering for a long time now, a systematic handling of the variability is still lacking adequate concepts for the associated software and model-side variability. This concerns in particular dependencies between variability in the control software and the simulation models. In the following

sections, the question is discussed, how to deal systematically with the occurring variability between the simulated production machines in software within the virtual commissioning[1]. Firstly, the paper gives a brief introduction into the Virtual Commissioning. In the following the systematic handling of variability in the context of virtual commissioning is discussed. Afterwards the different requirements for a so-called variant infrastructure within the virtual commissioning are mentioned and briefly explained. Finally a summary and an outlook about advancing the virtual commissioning with variant handling are given.

2 Virtual Commissioning (VC)

In contrast to the classical sequential development of machine and plant engineering, the virtual commissioning (VC) follows a different methodological approach. Parts of the commissioning are brought forward by using a virtual machine of the necessary system components. Thereby the software development can start early in the developmental process (Figure 1). Through the virtual commissioning the classic commissioning is not completely replaced, but it can be significantly shortened and simplified. By parallelizing the developmental steps and by utilizing the simulation feedback, VC significantly shortens developmental times, reduces developmental costs and improves product quality.

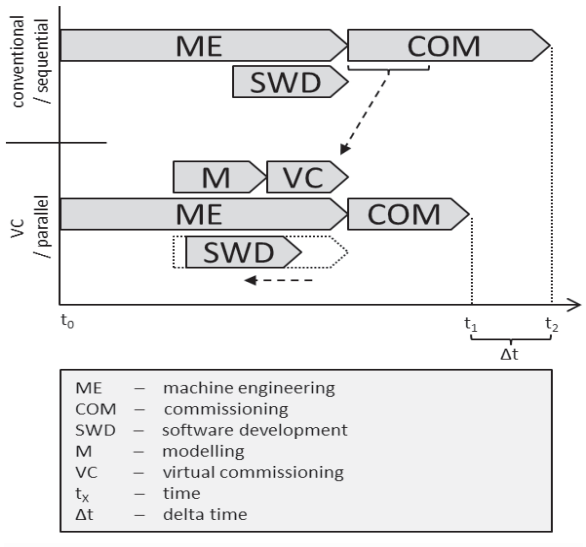

Figure 1: Concept of the Virtual Commissioning

[1] This work is funded by the Federal Ministry of Education and Research within the project Virtual Commissioning of Variant-Rich Systems (VivaSys) under the reference number 03FH085PX2.

Depending on the objectives of the VC, the components are displayed as simple as possible and as accurate as necessary in various domain specific simulation models. For example, a virtual machine tool is represented by kinematics and behavioral models. In this case the three-dimensional kinematics model shows the geometry, the kinematics and the collision calculation of the machine. Whereas the behavioral model describes the physical characteristics of the machine, such as the timing or the switching behavior. Simulation models can be connected to real control software via a (simulated or real) field bus system. The control software can be tested in this way at an early stage compared with the simulation model (Figure 2).

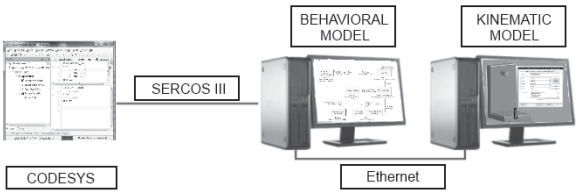

Figure 2: Integration of simulation models and control software

Usually there are different simulation tools for development and simulation of kinematics and behavioral models. Thus, for example the multibody simulation can be modeled in the simulation tool RecurDyn from FunctionBay [8] and the behavioural model can be modeled in SimulationX from ITI [9] or Simulink from The Mathworks [11]. The control software can be realized by the use of the developmental environment CODESYS [17] from 3S-Smart Software Solutions.

Simulation models, which have been developed in this way can be executed and synchronized at the same time as part of the co-simulation. Concepts that allow such coupling between simulation tools are, for example:

- The Functional Digital Mock-up (FunctionalDMU): An initiative of the Fraunhofer Gesellschaft with the aim to create a bridge between different simulations and visualization [3]. The runtime environment consists of a master simulator and simulators with appropriate wrappers that are connected to the master simulator interfaces, called slots.
- The Functional Mock-up Interface (FMI), which was developed in the context of MODELISAR

[12]. This concept was developed for the exchange of dynamic models, which enables the coupling of different simulation and modeling environments for cross-domain simulation [3].

These above concepts allow the exchange of information between simulation models during the simulation time. The focus of the following requirements is ensuring the consistency between simulation models before executing a possible simulation.

3 Handling of variability

Current simulation tools like RecurDyn, SimulationX or Simulink only support the modeling of individual systems and do not know concepts regarding variability. In industry, the current procedure in the context of VC is therefore that for each component individual simulation models have to be developed. Alternatively existing models are adapted by duplicating (so-called Clone and Own). In particular, the duplication can lead to increased maintenance costs and increased exdenditure of time throughout the software life cycle. As a change in one version may lead to changes in all other copies.

An economic use of VC requires inevitably a systematic observation and handling of variability within and between simulation models and associated control software.

A suitable concept for the systematic handling of variability and for the efficient production of highly individualized systems represents the product line engineering ([4], [6]). Instead of developing individual systems, which are independent of each other, the focus is in product line engineering from the outset on developing a set of systems that are associated with a particular application domain. This development is mainly done in two parallel processes: The domain engineering and application engineering. During domain engineering, a product line infrastructure for the systematic reuse of software artifacts is developed. This includes, for example, the analysis of variable requirements between systems in terms of features as well as a cross-system reference architecture or reusable implementation components. During the application engineering these artifacts are the basis, in order to generate specific members of the product line.

The generative software development builds upon the product line engineering. Goal of generative software development is to generate automatically highly customized and optimized systems from defined reusable components on the basis of a concrete system specification [7]. Core concept represents the generative domain model (Figure 3). This separates application-oriented concepts in the problem space from the concepts of the implementation in the solution space.

It allows separate development of domain concepts and reusable components and thus their individual modeling, implementation and evolution. Both models can be developed independently of each other in this way. The configuration knowledge maps the problem space to the solution space and represents the relationship between the two models explicitly.

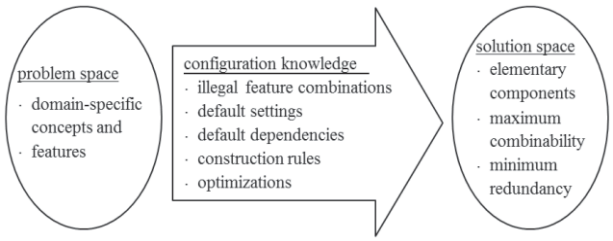

Figure 3: Elements of the generative domain model (see [7])

The product line engineering and the generative software development, thus, form a basis for the systematic handling of variability in the context of VC.

4 Requirements for variants infrastructure within the VC

Taken the product line engineering and generative software development as the basis for a software-based handling of variability, the following aspects have to be considered. These aspects are the essential requirements for a so-called variant infrastructure within the VC.

4.1 Development of feature models in the problem space

For a function-based approach of variant-rich embedded systems, feature models turned out to be successful ([2], [7], [5], [9]). In feature models variable requirements of a product line are managed and hierarchically structured in the form of features of the studied domain. The result is a comprehensive model of common and variable features and their dependencies between product variants. A distinction is made between mandatory and optional features, and (1..n):m-group relations that are realized by the feature types Mandatory, Optional, Alternative, and Or. Each fea-

ture can have a set of attribute-value pairs. Dependencies between features and attributes are defined by restrictions. For feature modeling, the tool pure::variants from the company pure-systems [13] can be used. Figure 4 shows, for example, the variable characteristics of a speed picker (using notation from [7]).

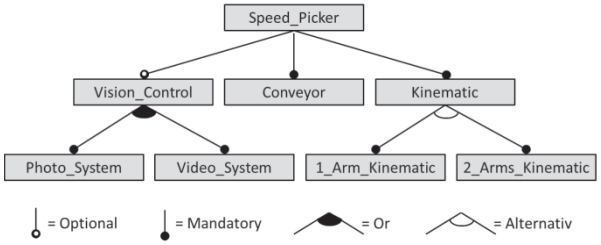

Figure 4: Extract of the speed picker feature diagram

In this feature tree, *Vision_Control* represents an optional feature, which could or could not be part of the *Speed_Picker* system. The *Kinematic* could be achieved by choosing one of two alternative features *1_Arm_Kinematic* or *2_Arms_Kinematics*. Finally, *Photo_System* and *Video_System* are Or-features. *Vision_Control* could be implemented either on the basis of using either one of them or both.

4.2 Development of reusable components in the solution space

In the context of VC, the solution space includes the control software and the simulation models. Therefore the question arises which aspects - such as technique, information, and procedure - have to be considered in the implementation of variability in control software and simulation models. In the following, essential aspects and the resulting requirements for the development tools are discussed in more detail.

a) Starting point for the implementation of variability in control software and simulation models is the variation point. This defines a separate, clearly identifiable area of the software or model, in which adjustments can be made for a specific system variant [1]. Variability is therefore clearly localized.

For the introduction of variability in control software and simulation models it is essential that the development tools contain language elements by which variation points can be realized. Here, the significant factor is, that these "model-specific" language elements differ from "regular" language elements:

- A variation point must be clearly visible for the developer for manual and graphical development. Eg by using a variant-specific color or a graphical/textual label of the variation point.
- A variation point must be clearly identifiable for automated processing. Eg via an annotation as a unique identifier. This is necessary for automated configuration and communication with the control software and the simulation models.

In addition, the potential language elements should ensure a uniform handling for realizing variability within the considered development tool.

b) The variation point essentially consists of a variability mechanism and the possible variants aside from a unique identification. The variability mechanism determines how a variation point is removed and is replaced by an associated variant. Figure 5 shows an example of a variation point in SimulationX.

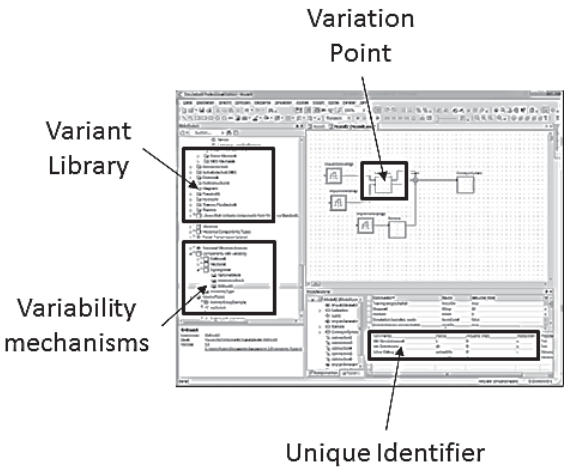

Figure 5: Variation Point in SimulationX

The development tools must provide mechanisms by which a variation point can be removed from the software or models in terms of configuration of a specific variant. Basically, the following mechanisms can be distinguished:

- During configuration a specific variant is selected from a set of predefined variants based on a specification. The variants are included in the control software and the behavioral models. Eg such variants could be selected by a signal routing.
- Within the substitution a variation point is replaced by a variant. The variation point specifies the condition under which this variant has to be inserted in the variation point. Such variants could be managed in the development

tool within a library or by a file system which is under a separate and external version control.

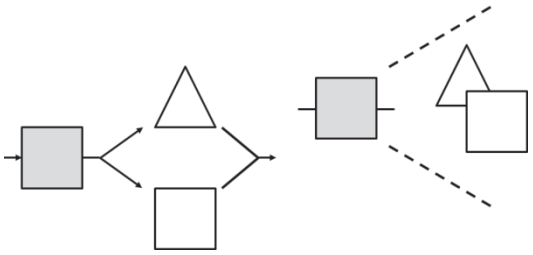

Figure 6: Variability mechanisms „selection“(left) and "substitution"(right)

The selection and substitution can be controlled via a specific set of variant configuration parameters. Meanwhile, a number of object-oriented development tools are offered for the modeling of control and simulation software. Therefore, the inheritance is another application for the selection and substitution of variants.

- The generation is based on a specification that eg may take the form of a blueprint. From this specification, the system variants will be generated by a generator.

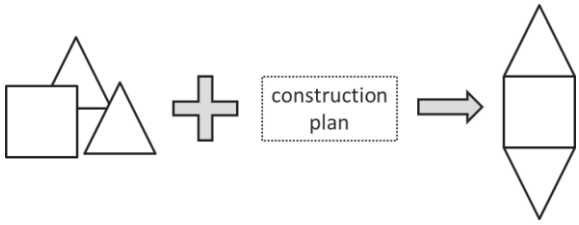

Figure 7: Variability mechanism "generation"

Not every concept is supported by each development tool. On the one hand, in the modeling of a variation point in a behavioral model, like SimulationX, it is possible to model all variants through signal routing, whereby a selection can be realized. On the other hand within the modeling of kinematic models, eg of multi-body simulation models in RecurDyn, only one variant may be part of the current model (see Figure 8). In this case, the variation point requires detailed information by which variants it may be replaced and where to find them.

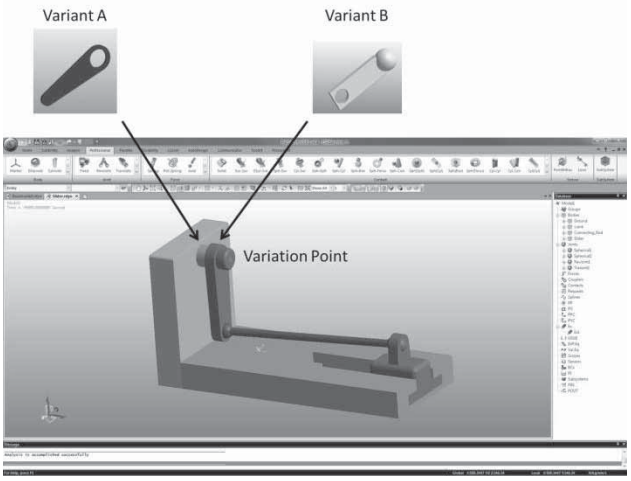

Figure 8: Modeling a variant-rich multibody model in RecurDyn

c) As part of the mapping of the problem space to the solution space, the features are mapped on variation points in the control software and the simulation models. Therefore, feature types *Optional*, *Alternative*, and *Or* have a strong influence on the realization of these variation points: Feature types have to be mapped to variability mechanisms. Experience shows that not every variability mechanism supports each feature type. Therefore, the tool side must provide potential variability mechanisms for the various feature types.

d) Depending on the size of the variant, that is associated with a variation point in the control software and the simulation model, different granularities of variability can be distinguished. The simplest form of variability is the so-called data variability. Variants describe the parameter values, which represent application-specific thresholds or characteristic curves. Another form of variability is the variant-specific signal routing within an application function. In the third form, variants consist of code blocks or model components that encapsulate variant-specific application functions. This enables the separate and possibly parallel development of functions. Such application functions may also be reused in a different context. In the development of control software by using CODESYS, a code block could be a class, a function, a function block or a set of parameters. For the relevant granularity the potential variability mechanisms has to be considered on the tool side.

4.3 Modeling of configuration knowledge

In the context of the VC the features from the feature model have to be mapped to variation points from the control software and simulation models by the configuration knowledge. For this mapping the knowledge about the features, the variation points in the control software and simulation models, as well as the dependencies between features and variation points (both among themselves and between problem and solution space) are required.

In the feature model dependencies between features can be specified as following: dependencies can depend on the position of a feature in the model hierarchy, the feature type or the special relations to other features. On the basis of the tool pure::variants it is possible to manage this knowledge.

With respect to the variability in the control software and simulation models various questions have to be answered:

- What comprises the knowledge to manage variability? Among others, this affects the information about the variation points, information about variants and information about the used variability mechanisms.
- Where to store this variability knowledge? Eg within the means of the development tools or outside the tools in a central repository.
- How to save the knowledge about the variability? This aspect relates to the structure of the data model and its implementation.

The mapping of features out of the feature model to variation points enables tracing of the distributed variation and automated configuration of the variant-rich control software and simulation models based on a feature selection. Here it is necessary to analyze the following aspects:

- What comprises the knowledge to manage dependencies? Eg the assignment of features to variation points of the control software and the simulation models, default settings, etc.
- Where and in what type the dependencies are to be saved? Eg within the respective development tools or in a central repository.

4.4 Synchronization using a variant manager

As part of the VC several software tools are working on the development of the control software and the kinematics and behavioral models closely together. Knowledge of the variability and dependencies within and between the control software, the simulation models and the feature model must necessarily be synchronized for the management and configuration of variant-rich systems. This is the central task of the so-called *variant manager* (Figure 9).

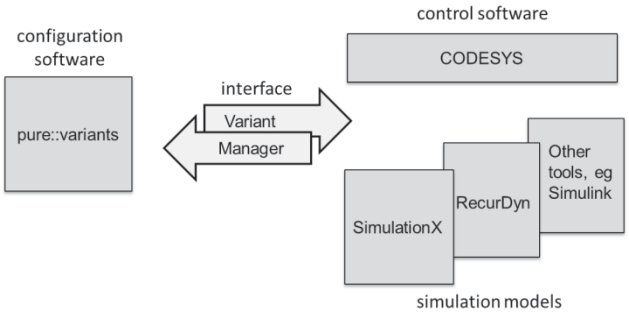

Figure 9: Variant infrastructure with variants manager

The variant manager provides the functionality to automatically share variability and dependency knowledge between different VC-development tools in the variant infrastructure. It ensures consistency of variability knowledge across the used tools. Due to different interfaces of the development tools it is up to the variant manager to provide an appropriate interface technology for the exchange of information. At the logical level the variant Manager must provide the following functionality to the development tools:

- To log on (and off) to the variant manager.
- To define the possible communication techniques at the technical level for integration into the infrastructure. This communication could be realized by the exchange of formatted files (XML, CSV, etc.), remote procedure calls, or an object broker.
- To consistently exchange the tool intrinsic variability knowledge on the basis of a defined set of operations.

Part of the development is to evaluate to what extent such a variant manager should be realized central in the form of an information broker or decentralized distributed in the VC-development tools.

5 Summary and Outlook

In the development of production machines, the virtual commissioning (VC) plays an increasingly important role. On the basis of virtual simulation mod-

els, the control software can be verified at a very early stage and can back flow simulation insights into the design of the machine. This leads to shorter development time and a higher product quality.

The challenge of the VC is to reduce the high costs of developing the simulation models. These costs can be significantly reduced by considering variability in the control software and the simulation models. A consideration of variability enables the systematic reuse of common parts of the model.

On the basis of product line engineering concepts, the requirements are analyzed for a variant infrastructure in this paper. This is the basis for a systematic presentation, management and configuration of variability in control software and simulation models. As part of the VC several software development tools work together. For a multi-tool synchronization of variability knowledge the requirements for a so-called variant manager are analyzed within the variant infrastructure.

The requirements have been determined as part of the BMBF-funded project Virtual Commissioning of Variant-Rich Systems (VivaSys). Currently a data model and the architecture for the variant infrastructure as well as a procedure for handling variability in the context of VC are developed.

6 References

[1] Becker, M.: Anpassungsunterstützung in Software-Produktfamilien. Dissertation, Technische Universität Kaiserslautern, 2004.

[2] Beuche, D.: Composition and Construction of Embedded Software Families. Dissertation, Universität Magdeburg, 2003.

[3] Blochwitz, T.; Otter, M.; Arnold, M.; Bausch, C.; Clauß, C.; Elmqvist, H.; Junghanns, A.; Mauss, J.; Monteiro, M.; Neidhold, T.; Neumerkel, D.; Olsson, H.; Peetz, J.-V.; Wolf, S.: The Functional Mockup Interface for Tool independent Exchange of Simulation Models. 8th International Modelica Conference 2011, 20-22 March 2011, Dresden, Germany, 2011.

[4] Böckle, G.; Knauber, P.; Pohl, K.; Schmid, K.: Software-Produktfamilien – Methoden, Einführung und Praxis. dPunkt Verlag, 2004.

[5] Clauss, M.: Untersuchung der Modellierung von Variabilität in UML. Diplomarbeit. Technische Universität Dresden, 2001.

[6] Clements, P.C.; Northrop, L.: Software Product Lines – Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley, 2001.

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

[8] FunctionBay GmbH: RecurDyn, www.functionbay.de/, 23.01.2014.

[9] ITI Gesellschaft für ingenieurtechnische Informationsverarbeitung mbH: SimulationX, http://www.simulationx.com/, 23.01.2014.

[10] Lee, K.; Kang, K.C.; Koh, E.; Chae, W.; Kim, B.; Choi, B.W.: Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice. In: Donohoe, P. (Edit.): Software Product Lines – Experience and Research Directions. Kluwer Academic Publishers, 2000.

[11] Mathworks: Simulink. www.mathworks.de/products/simulink/, 23.01.2014.

[12] MODELISAR (07006): Functional Mock-up Interface for Model Exchange. Document version 1.0, 2010.

[13] pure-systems GmbH: pure::variants Eclipse Plugin User Guide. 2012.

[14] Relovski, B.; Neumerkel, D.; Kleiner, N.; Welakwe, N.-A. S.: The MODELISAR Project and the Functional Mockup Interface. 1st Int. Conference on Multiphysics Simulation – Advanced Methods for Industrial Engineering, 22.-23. June 2010, Bonn.

[15] Wenk, M.: Virtuelle Inbetriebnahme. www.oth-aw.de/wenk/forschung/virtuelle_inbetriebnahme/, 03.01.2014.

[16] Wünsch, G.: Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme. Herbert Utz Verlag, 2007.

[17] 3S-Smart Software Solutions GmbH: CODESYS, http://www.codesys.com/, 03.01.2014.