doi:10.2478/acss-2014-0003
Applied Computer Systems
2014/15

Intellectual Model-Based Configuration Management Conception

Arturs Bartusevics[1], Leonids Novickis[2], Eberhard Bluemel[3]
[1-2] *Riga Technical University,* [3]*Fraunhofer IFF, Magdeburg, Germany*

***Abstract* – Software configuration management is one of the most important disciplines within the software development project, which helps control the software evolution process and allows including into the end project only tested and validated changes. To achieve this, software management completes certain tasks. Concrete tools are used for technical implementation of tasks, such as version control systems, servers of continuous integration, compilers, etc. A correct configuration management process usually requires several tools, which mutually exchange information by generating various kinds of transfers. When it comes to introducing the configuration management process, often there are situations when tool installation is started, yet at that given moment there is no general picture of the total process. The article offers a model-based configuration management concept, which foresees the development of an abstract model for the configuration management process that later is transformed to lower abstraction level models and tools are indicated to support the technical process. A solution of this kind allows a more rational introduction and configuration of tools.**

***Keywords* – Configuration management, configuration management model, model-based approach.**

I. Introduction

Software configuration management is one of the most important disciplines within the software development project, which helps control the software evolution process and allows including into the end project only tested and validated changes. To achieve this, software management completes certain tasks. The following configuration management tools are usually distinguished: identification of configuration units, unit version control and provision of parallel development, product constructions and installations, accounting and audit of configuration units [1], [2]. Nowadays there are numerous literature sources that describe the basic principles of solving the above-mentioned tasks. There are also many commercial and open source tools, which technically implement the principles described in literature [1]. Specialists in the field of configuration management note that successful introduction of configuration management initially requires awareness of the configuration management process within the particular project, abstracting from the use of concrete tools. The initial objective is to understand how the general tasks will be solved, taking into account the project specifics and general recommendations in the field of configuration management. Only after all of the general functions of the process have been identified, the most appropriate tools can be sought for the implementation of concrete functions [1], [2]. Despite the above-mentioned recommendations, in practice the activities for identifying the general configuration management process are skipped, and process introduction already begins with the installation of tools. Within the software development project tools are installed with an aim to provide technical support for certain activities rather than the process on the whole. For instance, several developers begin writing a software code, which is reduced to correcting the same source code files. In order to detect who has made the changes and what these changes are, a version control system is installed. For a period of time, programmers can successfully make changes, but soon a client is given the first product version of the acceptance test. The client detects errors and asks to correct these errors within two weeks, however, at the same time, work is started on a new model for the same product and the expected time of completion is approximately one month. This requires making changes to the same code. As a result, a parallel branch is created in the version control system, in which changes are made in line with the new module. In the initial branch, errors identified by the client are corrected and the new version is supplied to the client. After a while, new module functionality has to be supplied, which leads to a necessity to merge two parallel source code branches. Given the fact that no merging strategy had initially been defined, errors emerged in the product construction, which had already been corrected in the previous version. Repeated corrections had to be made. After the error was eliminated, a strategy for parallel development was never discussed. What would happen in case of another necessity to perform parallel developments? Most likely the same errors would have to be corrected. This example describes only the issues concerning version control and parallel development, yet other configuration management tasks can be similarly discussed as well. The mentioned example allows concluding that without overseeing the total process, a solution to configuration management problems is reduced to fixing concrete tools or working with the source code rather than improving the process. As a result, the development project involves one or two people, who after a certain period of time eliminate particular errors by correcting tool configuration or code. In the course of time, there are so many changes to be made that only one particular person or a group of people are able to understand and oversee the process. To make changes, it is necessary to have an irrationally large amount of time. If changes have to be made by a person who previously has not been involved in the project, this becomes nearly impossible, for there are numerous changes, tools and configurations, yet lack of a general picture of the process or relevant documentation [1], [2], [8].

Problems in configuration management also emerge if initially the process is defined on the whole. Let us assume that the person in charge defined the configuration management process by abstracting him- or herself from the introduction of concrete tools. These tools are chosen and installed afterwards by following particular needs. The process functions until the necessity emerges to change something or to correct an error. Oftentimes due to a lack of time or resources, changes are made directly to the tool configuration of the code, yet the general picture of the process or documentation is never updated. This leads to a situation in which after a period of time the process description no longer corresponds to the actual functioning and configuration of tools. The above-mentioned issue becomes relevant again, namely, changes to configuration or to the code become irrational, for they do not prevent causes and, therefore, imperfections in the process [1], [8].

In order to solve these problems, a tool or a method is required that would relatively quickly help obtain a configuration management process model, which in turn would be able to react upon changes in a particular project problem environment. One of the most important tasks is to find an intellectual configuration management solution. In the context of this article, the term 'intellectual' relates to the ability of a configuration management model to react upon changes in a problem environment. The reaction should be expressed in a way for a configuration manager to see a particular place within the process model where changes need to be made in order to adapt to the problem environment or to correct an error. Thanks to this, a configuration manager can make timely changes to the process model and the tool configuration. This can potentially prevent irrational fixing and configuration of tools, as well as help prevent problem causes more effectively.

II. Related Research

Initially model-based solutions began to emerge in cases where only one of the general tasks of configuration management was being solved, for instance, identification of configuration units, version control or building [9] - [12]. In the publication [9], the authors offer to perform modeling of the product to be developed. In the beginning, it was emphasized that by defining the potential candidates for configuration units not only the source code, but also documentation needs to be taken into account, describing different aspects of each product. The modeling approach is based on a gradual creation of a system branch-like structure. Initially documents, models and sub-systems of the product to be defined are developed. In the next iteration, documents are divided into groups and links between documents are defined. Sub-models are defined for models and their mutual links are described. Components of sub-systems are defined, for instance, databases, graphic user interfaces, etc. In the following iterations, the structure is broadened with new elements and their mutual links. The process continues until a decision is made that each lower layer top cannot be subdivided and all links between defined elements at the given moment are determined.

When studying version control and parallel development task solutions, model development can also be observed [10], [4]. The general idea in the study [10] is an algorithm, which creates a metamodel for the controllable units. This metamodel describes the syntax of a model, which might be subject to version control. The method is foreseen for version control systems, which control models rather than the source code, which is characteristic of projects where a model-based approach is applied. Thanks to the metamodel, it is possible to not only manage model changes, but also to perform merger of various model versions and to observe conflicts during the merger similar to how this can be observed when joining together source code file changes. It lays out the main problem that contemporary version control and parallel development can be limited to source code file control. Projects where a model-based approach is used have the necessity to control versions for models; however, in separate projects, due to their specifics, the existing version control systems are unable to offer solutions for all the tasks. For this reason [4] source authors offer a methodology, which foresees development of an abstract model version for the control system. Gradual transformation of the obtained model into a lower abstraction level model may obtain transfer for a new version control system development.

Although the product building process in practice usually is associated with the choice of particular compilers or tools, there are solutions, wherein the mode-based approach is used – initially an abstract building model is created and only afterwards tools are introduced that implement the model in practice [11], [12]. The publication [11] speaks not about particular tools, but about an abstract model, which describes the building and installation process. As a solution, a model is offered that describes activities to be performed in the building and installation process. Each activity provides a summary of tools that are used to technically implement and to automate activities. In its turn, another publication [12] emphasizes the problem of a qualitative product building absolutely requiring the right choice of configuration units, which should be part of the building. The publication [12] offers model-based methodology, which determines the cluster of configuration units that should be part of the building. The methodology makes use of heuristics and analyses all possible configuration unit clusters, which potentially might be included into the building. A version is chosen providing the best possible product configuration, for example, in the context of the performance.

Speaking of configuration management model-based solutions, which encompass several configuration management tasks, it must be concluded that there are not many of them. Only a few such solutions were found, in which a more or less complex configuration management process would be emphasized in the context of several tasks [3], [5], [6]. In order to find a better view of the maturity and disadvantages

of the solution in the course of the study, in parallel some overarching principles of the model-based approach were studied [7], [14]. A functioning model-based solution should have the following traits:

- Existence of metamodels. The solution must have a source from which a configuration management process model can be developed.
- Models of different abstraction models. In order to focus upon the process and not on tools, the model-based solution must have an abstract model, which describes the configuration management process irrespective of the computation and tools that will be used in the process implementation. Yet there should also be other types of models, in which computation traits, platform specifics and the role of tools within the process can be reflected.
- One of the mandatory components of the solution should be a link between different types of models and transformation rules, which allows transforming a model with a certain level of abstraction into a model of another level of abstraction. Transformation rules should be formalised to be understood by a computer.
- Tool support. In order not to leave the solution solely on the theoretical level, there should be tools that can create models from a ready metamodel and perform the necessary transformations.
- Link with the problem environment. A link is needed for the solution to be sustainable. Otherwise, the developed and implemented model is unable to react upon changes in the problem environment and there will be no possibility to determine the weak points of the model. In the course of time, this may lead to changes to the configuration of certain tools or code changes and without overseeing the process the process model may lose its relevance.

In the course of the study, no single solution was found that would meet all of the above-mentioned requirements. As part of the solution [3], a unified conception was created for configuration management and model-based development – a metamodel that allows creating an abstract product configuration model, a tool based on Eclipse Modeling Framework, which allows obtaining a specific configuration model from an abstract model platform, as well as instructions on how to broaden methodology and implement a tool. In the course of research, in this publication [3] it had to be concluded that although the solution corresponds to the general principles of the model-based conception, it is more oriented towards project, the development of which is based on the model-based approach. A metamodel exists only for the task of configuration unit identification, yet there are no concrete indications or recommendations as to how this can be applied to other configuration management tasks. There is also no link with the project problem environment; therefore, it is uncertain as to the most rational ways of making changes to the model if such a necessity would emerge.

As opposed to the solution above [3], which puts an emphasis on the identification of configuration units and detection of dependents, the source [6] offers a methodology that looks at the configuration management process on the whole. Configuration management principles for this solution are taken from the ITIL (Information Technology Infrastructure Library) standard and later on abstract models are created, from which an abstract configuration management process can be developed, and later this model is transformed into a platform-dependent model. This approach makes use of the general principles of model-based development. Models based on metamodels offer the necessary abstraction, which improves the management of configuration process, the monitoring of it, and in case of necessity, allows users to implement the model for a certain technology, for instance, when performing the model transformation. A system prototype is created, which implements the model-based configuration management. The author of the study [6] projects further large-scale research in order to come to a particular solution, which is also explained in the conclusion. Although the given solution also includes the implementation for the offered model-based configuration management, the solution is orientated towards a single type of technology (JAVA) and no particular details of implementation have been revealed. Therefore within this solution, it is hard to assess the possibility of its practical use and nothing is mentioned about the link with the problem environment.

By analyzing configuration management solutions within the more or less model-based approach, a solution was found [5], which does not just explore the configuration management process on the whole by listing all of the general tasks, but emphasizes that mutual integration of various configuration management tools plays an important role. In order to maintain a full configuration management process, several tools are required: version control systems, problem management systems, building servers, continuous integration servers and many other tools. In practice, all of these tools work independently of each other. To ease the configuration management process, an approach is offered to integrate all of these tools into clusters. However, to integrate various configuration management tool clusters, a general concept of each tool to be integrated should be defined [5]. The publication offers ontology of tasks for the configuration management processes. This ontology is used as a configuration management model, which shows in what ways various configuration management tools will be integrated. The ontology is largely based on change control, which is one of the main concepts in configuration management. The ontology offers information about mutual links within the configuration management sub-process, which is expressed through concepts, links, tasks, agents and entry data clusters [5]. It must be acknowledged that the publication has no concrete indications on how the ontology can be applied to configuration management of a particular project. Speaking of a link between the problem environments, it is unclear how changes can be made to ontology, what kinds of ontology

editors are suggested to use and how to determine the moment when changes should be made.

Assessment of the existing intellectual solutions of configuration management did not result in finding solutions, where one of the artificial intellect methods would be applied to the configuration management process in general. Even if any of the existing solutions could be considered intellectual, it is only directed at a single task of configuration management. An example of this phenomenon is the source [13], in which the identification task of configuration units is being solved. The solution foresees application of fuzzy logic theory to create a multi-criteria decision-making system. The main goal of decision-making is to determine an optimal cluster of configuration units, which can then be subjected to the configuration management process [13].

III. General Overview of the Intellectual Model-Based Configuration Management

The general aim of the intellectual model-based configuration management is to create a configuration management process model for a particular project and to ensure a dynamic link between the configuration management problem environments and to develop a model. Initially, a computing-independent model is created from the configuration management metamodel, which only conceptually describes the solution of general configuration management tasks mentioned in the introduction of the article. Later with the help of transformations, the computing-independent model is transformed into a platform-independent model. This model already shows what tools might be necessary for the technical support of the process and how these would exchange information. However, nothing is mentioned about particular tools, platforms, development technologies, etc. For instance, this model mentions the version control system and the type of information it passes on to the building server, yet nothing is said about the version control system. Within the platform-independent model it is the information exchange and not the concrete system that is important. At this stage, for instance, it is not important whether the version control system is Subversion or Git. The platform-independent model is being transformed to a platform-specific model, which already indicates concrete tools, development technologies, platforms, etc. At this level, for example, it can be seen how the Subversion version control system should be configured and what information cluster it passes on to the Jenkins continuous integration server in order to implement an initially planned configuration management process. At the moment when a configuration management process model can be obtained from the metamodel step by step by using transformations, an expert system is created, which ensures a link between the model and the configuration management problem environment. For this very reason, the word "intellectual" forms part of the solution title. The expert system collects information from tools, which technically ensure configuration management. The expert system knowledge base contains two kinds of rules. The first group of rules, depending on the obtained information in the problem environment, will determine in which of the models changes should be made. The second group of rules works with metamodel components. If one of the components is incomplete or cannot ensure the necessary changes in the model, recommendations for the metamodel updates are being formulated. At this given moment it is unclear to what extent this link and the expert system activity will be automated. The authors hope that the necessary answers will be given by experiments planned in the future. Figure 1 shows a scheme for intellectual model-based configuration management.

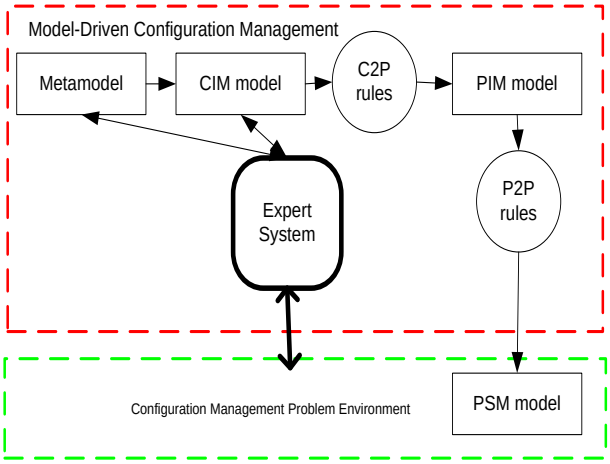

Fig. 1. Conception of model-based configuration management.

IV. Intellectual Model-Based Configuration Management Components

Metamodel – a source from which components are taken for the development of the configuration management model. A metamodel contains components that have certain syntaxes, semantics and principles of development. Each component has a description, which includes a clarification in what ways it is to be linked with other components. A metamodel has a link with the expert system, which collects information from the configuration management problem environment and indicates components, which need to be changed or indicates a necessity to create new components to better develop a process model.

CIM – a computing-independent model for the configuration management process within a certain development project. Model components are taken from the metamodel. The CIM is linked to the expert system, from which information about the necessity to conduct a certain type of changes is received from time to time, and this is linked to changes in the problem environment or error elimination.

C2P rules – transformation rules, which determine how to obtain PIM from the CIM (computation-independent model). It is expected that rules will contain information on the kind of tools required to technically support modelled configuration management processes and how these tools will mutually exchange information.

PIM – configuration management model, which thanks to the C2P transformation rules can be obtained from the CIM. As mentioned earlier, the model contains information about tools and their mutual information exchange, yet nothing is said about concrete technologies or platforms. At this stage, for instance, it is not important whether the version control system is Subversion or Git; however, the number of parallel development branches and the order of their integration play a significant role.

P2P rules – transformation rules, which determine how a PSM (platform-specific) model can be obtained from the PIM. The rules supplement the PIM with information on particular tools and platforms. Presumably, rules will be corrected by a user depending on which exact technologies this user would choose to use. For instance, seeing within the PIM the way version control is organized as well as parallel development branches, the configuration manager will be able to choose a version control system, which could implement the requirements of the model and define within P2P rules whether this is a Subversion, Git, Mercurial, or another version control system.

PSM – specific model of the configuration management process platform. This model can be obtained through transformation from the PIM. Particular technologies already form part of this model and the tools mutually exchange information. The model shows what kinds of tools need to be installed, configured and what information exchange interfaces must be ensured to implement a modelled configuration management process.

Expert System – the part of the solution, which ensures the link between a model and its problem environment. The system consists of a module that collects information from the installed tools and rule base of configuration management. The rule base defines rules, which depending on the obtained information, are able to determine at which model stages changes need to be made. If, for instance, the tool logs show that activities do not correspond with the process, the expert system indicates the inconsistent place in the model and offers to make changes. The rule base also contains rules that on a certain level can control the metamodel. Namely, if the existing components of the metamodel are unable to ensure the necessary changes in the model, the system offers to update the metamodel. It is expected to supplement the rule base with rules, as well as to correct the existing ones.

V. Expected Gains from the Offered Solution

The main expected gain from the offered solution is the possibility to have an overview of the general configuration process. Thanks to the fact that the PSM provides information on the type of tools that are necessary and the way of exchanging information by these tools, there is a possibility to automate the process. The model allows configuring tools in accordance with aims of a particular process instead of correcting a single error. This allows a more effective use of resources for tool configuration and instead of eliminating a particular mistake, allows getting through to its cause by minimizing the chance for the error to repeat itself in the future. Another gain would be to make the process more transparent. If a new person joins the project, he or she can become acquainted with the configuration management process on the whole and understand in what ways and with what purposes the tools function. Transparency of the process allows, in case of necessity, making changes to tool configuration more effectively. Such a solution would prevent from wasting a lot of time reading different types of documentation and studying the source code. At the same time, the expert system would continuously maintain a link with the problem environment and help configuration managers timely react to changes or errors in the process. This would lessen the probability that a process model would lose its relevance over time. The main goal of this solution is to improve the quality of configuration management process, raise tool configuration effectiveness and decrease maintaining costs at the expense of having more possibilities to prevent error causes rather than the error itself.

VI. Conclusion and Further Research

The article offers the concept of intellectual model-based configuration management. At the given moment, none of the solution components is being implemented in practice. Therefore, there is a long way to go to the practical application of this solution. One of the most important further studies will have to do with the development of a configuration management metamodel. The authors believe that the creation of a metamodel is one of the main factors, which will determine whether the end result is successful, since, without a qualitative metamodel, no full-fledged model can be created for the configuration management process. At the given moment it is assumed that a qualitative metamodel is the one where the principles, syntax and semantics of component creation are clear. If the general principles of the metamodel are clear, in case of necessity it will allow modifying the metamodel and increasing the probability that the model will be considered by specialists of the field because there is a phenomenon of difficulty to accept and to apply new and complex things. In other words, a complex and hard to understand solution will be perceived with care and the obtained results are not likely to be trusted. For this reason, it is necessary to study contemporary popular modeling notation and to assess the most suitable configuration management metamodel components. Components must be developed based on the idea of being easily perceived visually, i.e., to make it easier for users to understand the essence of components and the potential links with other components.

The next step is based on the created metamodel to conduct experiments, through which different Computing Independent Models for configuration management processes will be developed in various projects. One of the main goals of experiments would be to assess the ability of the metamodel as fully as possible, by developing a computing-independent model for the configuration management process. Further research will largely depend on experiment results and at this

moment these results are hard to project. It can be added that depending on the way how this model-based solution will be developed, one of the most crucial stages will be the development of an expert system to maintain a model link with the problem environment. In the process of developing the expert system, tasks like problem environment monitoring and principles of drawing up rules will have to be solved. Presumably, this would be an independent study in terms of the offered solution in the context of this article.

ACKNOWLEDGEMENTS

The research has been funded by the ERDF (ERAF) project No.2011/008/2DP/2.1.1.1.0/10/APIA/VIAA/018 "Development of Insurance Distributed Software Based on Intelligent Agents, Modeling, and Web Technologies".

REFERENCES

[1] R. Aiello, Configuration Management Best Practices: Practical Methods that Work in the Real World (1st ed.). Addison-Wesley, 2010.
[2] A. Berczuk, Software Configuration Management Patterns: Effective TeamWork, Practical Integration (1st ed.). Addison-Wesley, 2003.
[3] W. Pindhofer, Model Driven Configuration Management. Master work of Wien University, Wien, 2009.
[4] T. Buchmann., A. Dotor, B. Westfechtel, Model-Driven Development of Software Configuration Management Systems. ICSOFT 2009 - 4th International Conference on Software and Data Technologies 2009.
[5] R. Calhau, R. Falbo, A Configuration Management Task Ontology for Semantic Integration. Proceedings of the 27th Annual ACM Symposium on Applied Computing Pages 348-353 ACM New York, NY, USA, 2012. http://dx.doi.org/10.1145/2245276.2245344
[6] H. Giese, A. Seibel, T. Vogel, A Model-Driven Configuration Management System for Advanced IT Service Management, http://www.hpi.unipotsdam.de/giese/gforge/publications/pdf/GSV-MRT09_paper_7.pdf.
[7] J. Osis, E. Asnina, Model-Driven Domain Analysis and Software Development: Architectures and Functions. IGI Global, Hershey - New York, 2011, 514 p.
[8] Y. Udovichenko, Upravlenie projektami, http://experience. openquality.ru/software-configuration-management/, 2011.
[9] Object-Oriented Software Engineering Using UML, Patters and JAVA "Software Configuration Management, http://www.bilkent.edu.tr/~bakporay/cs_413/Bruegge_L28_ConfigurationManagement_ch12lect1.ppt.
[10] K. Altmanninger, Models in conflict - towards a semantically enhanced version control system for models. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2008;5002 LNCS:293-304.
[11] P. Sindhuja, N. Surajit, Software Deployment: Concepts and Technologies. ICFAI Journal of Systems Management, 2008.
[12] O. Bushehrian, Automatic object deployment for software performance enhancement. The Institution of Engineering and Technology 2011, Vol. 5, Iss. 4, pp. 375–384, 2011.
[13] Juite Wanga, Yung-I Lin, A fuzzy multicriteria group decision making approach to select configuration items for software development. MathematicsWEB, Fuzzy Sets and Systems, 2002.
[14] O. Nikiforova, N. Pavlova, K. Gusarovs, O. Gorbiks, J. Vorotilovs, A. Zaharovs, D. Umanovskis, J. Sejans, Development of the Tool for Transformation of The Two-Hemisphere Model to The UML Class Diagram: Technical Solutions and Lessons Learned. Proceedings of the 5-th International Scientific Conference „Applied Information and Communication Technologies", 2012, Jelgava, Latvia, pp. 11-19.

Arturs Bartusevics is a doctoral student at Riga Technical University, the Faculty of Computer Science and Information Technology, the Institute of Applied Computer Systems. He obtained BSc (2008) and MSc (2011) degrees in Computer Science and Information Technology from Riga Technical University. His research areas are software configuration management, release building and management process and its optimization. He works at Ltd. Tieto Latvia as a Software Configuration Manager.
E-mail: arturik16@inbox.lv

Leonids Novickis is a Head of Division of Applied Systems Software. He obtained Dr.Sc.ing. degree in 1980 and Dr.Habil.Sc.ing. degree in 1990 from the Latvian Academy of Sciences. Since 1994 he has been regularly involved in different EU-funded projects: AMCAI (INCO COPERNICUS, 1995-1997) – WP leader; DAMAC-HP (INCO2, 1998-2000), BALTPORTS-IT (FP5, 2001-2003), eLOGMAR-M (FP6, 2004-2006) – scientific coordinator; IST4Balt (FP6, 2004-2007), UNITE (FP6, 2006-2008) and BONITA (INTERREG, 2008-2012) – RTU coordinator; LOGIS, LOGIS-Mobile and SocSimNet (Leonardo da Vinci) – partner. He was an independent expert of IST and Research for SMEs in FP6 and FP7. He is a corresponding member of the Latvian Academy of Sciences and an elected expert of the Latvian Council of Science. His research fields include Web-based applied software system development, business process modeling, e-learning and e-logistics.
Ė-mai:l lnovickis@gmail.com

Eberhard Bluemel, Dr. rer.nat. Dr. h.c., is a Head of Fraunhofer IFF EU Office. Since 1992 he has been regularly involved in different EU funded projects as a project coordinator. His fields of applied research projects are associated with themes of Operation Research, e-Logistics, Software Management, Virtual Reality and Training.
E-mail: eberhard.bluemel@iff.fraunhofer.de