UNIVERSITÄT DORTMUND

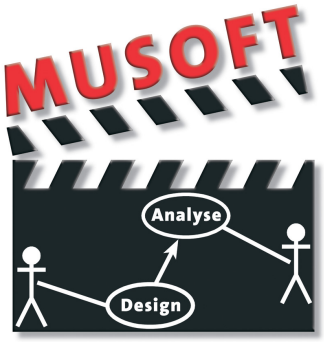
Fachbereich Informatik
Lehrstuhl für Software-Technologie

M E M O Nr. 137

MuSofT Bericht Nr. 3

A Framework for Lightweight Object-Oriented Design Tools

Klaus Alfert Jörg Pleumann Jens Schröder



April 2003

Internes Memorandum des
Lehrstuhls für Software-Technologie
Prof. Dr. Ernst-Erich Doberkat
Fachbereich Informatik
Universität Dortmund
Baroper Straße 301

D-44227 Dortmund

ISSN 0933-7725

A Framework for Lightweight Object-Oriented Design Tools*

Klaus Alfert Jörg Pleumann Jens Schröder

Lehrstuhl für Software-Technologie
Fachbereich Informatik
Universität Dortmund

Abstract

Teaching object-oriented software engineering and design with industrial tools is not always satisfying, since the focus of these tools lies on development rather than education. In this paper we present a family of graphical tools for object oriented design devoted explicitely to teaching. They are built upon a common framework and a metamodel approach.

1 Introduction

Teaching object-oriented software engineering is becoming more and more part of the basic curriculum for computer science, similar to object-oriented programming some years ago. Learning software engineering requires that – in addition to studying the concepts behind languages, formalisms and methods – the students work actively with these languages, formalisms and methods (similar to learning programming). Hence, tool support for the various notations is desirable. However, typically used industrial tools, such as Rational Rose or Together, have significant drawbacks in an educational setting.

These drawbacks arise from the fact that professional tools are rather "heavyweight" pieces of software, both in terms of their feature set and the hardware required to run them smoothly. Using such professional tools, there is a high risk that merely the tool handling is taught instead of focusing on the particular language or method. If different tools for different notations are used, this situation becomes even worse, since the students have to be familiar with all these programs before working effectively with the supported notations. The tools' user interfaces are quite complex, with lots of features aimed at efficient development in industry. Unfortunately, too many features are distracting when using the tool for demonstration purposes, e.g. in a lecture hall. For educational purposes it is sometimes required to use a simplified notation. This requires the tools to be easily adaptable, which is usually not the case. Last but not least, commercial tools always have the problem that it might be costly for the students to use them at home.

All these deficiencies led to our new approach. In the MuSofT project (multimedia for software technology, [DE02]) eight German universities work together on teaching software engineering with the help of multimedia technologies and tools. The projects at Dortmund are concerned with teaching software architecture based on an object-oriented notation and with teaching process modeling using the Unified Process. In both cases, we need tools for the graphical notations. The experiences of another MuSofT project [Kel02] showed that simplified CASE tools for didactical purposes are a promising approach. Therefore, we decided to develop our own suite of design tools, which are presented in this paper. We have termed these tools "lightweight" to distinguish them from the heavyweight professional ones.

The rest of this paper is organized as follows: In section 2, we present the tools from a user's point of view. In sections 3 and 4 we discuss the framework and the metamodel which provide the basis for the tools. In section 5 we report on some experiences, followed by an outlook on future work in section 6.

*This work is supported by the German Federal Ministry of Education and Research, grant 08NM098.

2 Lightweight Design Tools

For the MuSofT projects at Dortmund we have currently developed three lightweight tools for teaching and learning specific aspects of object-oriented software engineering:

- an editor and simulator for UML statemachines,
- an editor for software architectures (reusing parts of the former tool), and
- a process modeler for the Unified Process.

The tools can be used both for presenting examples of specific object-oriented design techniques during a lecture and for training the application of these techniques during exercises. A detailed discussion of elements and features common to all tools is given in section 3.

2.1 Statemachine Editor

One task of the design process is to specify the behavior of the software that has to be developed. In object-oriented systems statecharts [Har87, HN96] are a well-known technique for that purpose. For teaching and learning this technique a statemachine editor (see Fig. 1) was developed, incorporating the UML-notation for statecharts [OMG03]. It offers different components for drawing (e.g. a toolbar, an inspector, or a treeview) and presenting (e.g. a fullscreen mode) statechart diagrams.

To give a better impression of the modeled behavior of a software system, the statemachine editor provides a simulation component that can execute a statemachine. The simulation component evaluates the drawn diagram and highlights active states. Through a suitable graphical interface the user has the possibility to simulate events that trigger the transitions between states of the modeled system.

2.2 Software Architecture Editor

Another task of object-oriented design is modeling the architecture of the software which has to be created. The software architecture editor (see Fig. 2) is a tool that allows modeling software architectures, covering both static and dynamic aspects of these architectures. The editor employs a notation based on UML-RT [SR98] which, in turn, goes back on Real-Time Object-Oriented modeling (ROOM) [SGW94]. The dynamic aspects of each architectural component are again described using statemachines. Similar to the statemachine editor, the software architecture editor provides different drawing tools and presentation modes.

A simulation component animating the behavior of the modeled software architecture is also part of the tool. To illustrate the behavior of the architecture, the statechart diagrams are evaluated and visualized analogously to the simulation component of the statemachine editor. Yet, in this case, a potentially large number of statemachines is simulated in parallel.

2.3 Process Modeler for the Unified Process

To manage the development of software various process models exists. A process model particular suited for developing object-oriented software is the Unified Process [JBR98], a generic and iterative process model, which has to be adapted to the project's requirements before use. The process modeler tool (see Fig. 3) allows tailoring and instantiation of the Unified Process for a project in a graphical manner.

The editor generates a XML-representation of the instantiated process which can be used by other tools.

3 Basic Framework

To avoid implementation of each of the above applications individually and from scratch (thus reinventing the wheel over and over again) and to also establish a common look and feel for all of them, a Java/Swing-based framework for lightweight modeling tools was developed. The framework provides two main components: A more-or-less fixed user interface including an application frame, and a variable metamodel underneath.

To be suitable for educational purposes, the user interface was designed to be simple, intuitive to use and non-distracting. The largest part of the screen is occupied by the model itself, because this is the application's main focus. All graphical aspects of modeling, like moving or resizing elements or drawing graphical connections between them, are handled here. It is not necessary to use nested menu structures for common tasks like loading, saving, printing, or adding new elements to the model. Instead, these everyday functions are accessible from

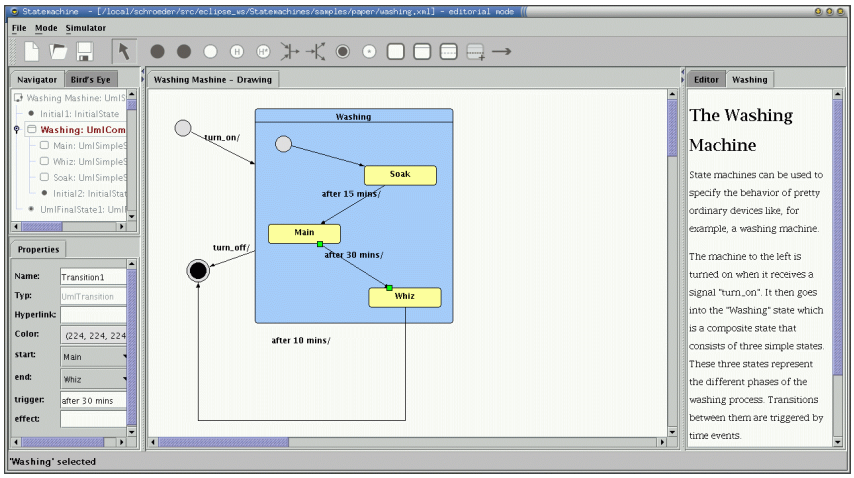


Figure 1: The state machine editor for visualizing UML-statecharts

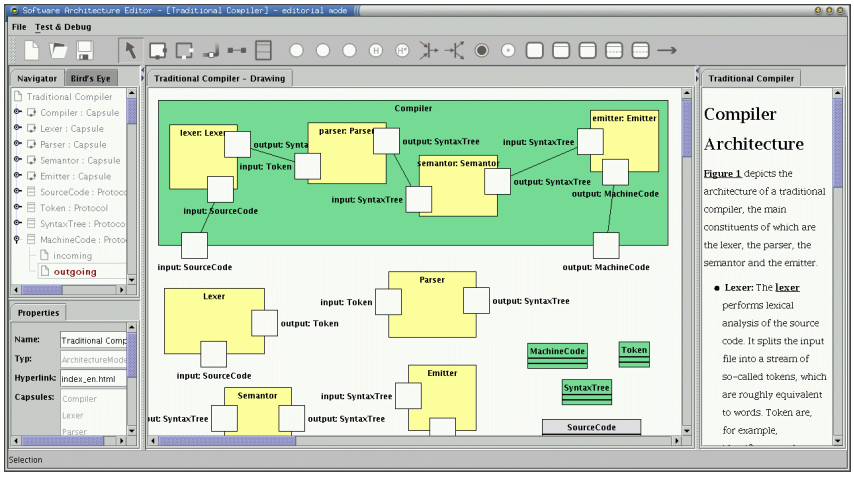


Figure 2: Example of a compiler architecture modeled with the software architecture editor

a toolbar atop the diagram area, with clear graphical icons describing each function.

In addition to the main diagram area and the toolbar, the application provides a number of user interface components with more specific purposes:

- The Navigator shows a tree-like representation of the model (based on the nesting of elements) and allows quick selection of single elements, which results in having them focused and selected in the diagram area.

- The Inspector displays the properties of the currently selected model element and allows their modification. It is particularly useful for those properties that have either no direct graphical representation or cannot be modified inside the drawing area for other reasons.

- The Bird's Eye gives an overview on the positions of all model elements and is useful for models that span more than a single screen.

These three components are placed left to the diagram area, honoring the common principle of placing navigational items on the screen's left side. All three can easily be hidden when only the diagram itself is of interest, for example during a lecture.

On the opposite side, that is, to the right of the diagram, a hypertext viewer and editor can be shown if required. This component usually comes into play when the students are using the tool at home. It allows annotation of a model: every element may be given a hyperlink to such an annotation. When the element is selected in the drawing area, the corresponding hypertext is displayed on the right hand side. Additionally, specific hyperlinks in the hyper-

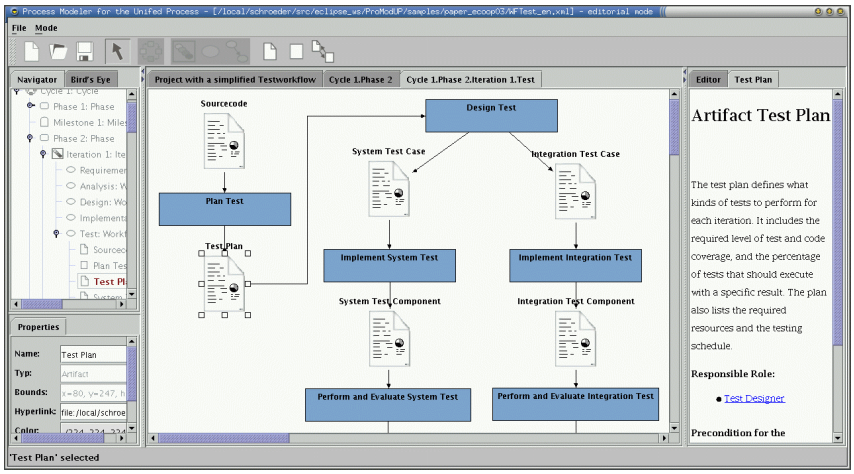


Figure 3: The process modeling editor for tailoring the Unified Process

text allow focusing on individual model elements or even modification of the model. This allows provisioning of complex, hypertextually annotated models which the students should study. Another area where the hypertext becomes handy are assignments: the students are presented a hypertextual assignment, and they can annotate their solution using the same means. The solution is submitted in digital form, and a tutor uses just the same tool to review it.

Developing the user interface parts of the framework has been eased by using JHotDraw [Kai01], a class library for applications dealing with technical drawings. JHotDraw builds upon the notion of a *drawing* (displayed in our diagram area) which my contain an arbitrary number of so-called *figures* (graphical representants of single model elements). Interestingly, although JHotDraw originated as an attempt to demonstrate the use of design patterns [GHJV97], it does not have a clean separation between model and view (in the sense of MVC), that is, there is no real model underlying the figures. Thus, it became necessary to extend the basic JHotDraw figures in a way that they rely on a proper (meta-) model underneath.

4 Meta Modeling

The aforementioned user interface provides different views of a model whose syntax and semantics are determined by an application-specific metamodel. Apart from the figures that make up the graphical representation, this metamodel is the part where the various possible modeling applications differ the most. It is also the part that requires the largest implementation effort, so it makes sense to strive for a generalizable solution here.

The solution we chose is based on the notion of a generic metamodel, or, practically speaking, a box of building bricks from which application-specific metamodels can be built. It is loosely based on ideas also found in the Meta Object Facility (MOF) [OMG02] or similar metamodeling approaches. Yet, it is not an implementation of the MOF nor is it a true meta-metamodel in the strict sense, because elements of the application-specific metamodel are derived from the generic metamodel via inheritance rather than instantiation.

At its very core, the generic metamodel consists of a graph structure with (nestable) vertices representing metamodel concepts and edges forming relationships between concepts. Each element can be augmented with an arbitrary number of uniquely named attributes and association ends. Attributes are used for storing primitive values. The currently supported types for these attributes are Boolean, Integer, String, Color, and Date. Examples for metalevel attributes are the visibility specifiers (public, private, ...) found in class diagrams. Association ends provide anchor points for associations, which are used to link elements of the metamodel. They may have an inverse and a multiplicity greater than one. They are first class members of the metamodel, so the implementation can easily keep both of their ends consistent. Associations differ from edge elements in that they usually do not have a graphical representation.

The generic metamodel elements also provide a number of methods used for enforcing the syntax rules inherent in the supported modeling language. For example, both the vertex and the edge elements

have methods named canConnect(), the result of which determines whether an edge element can connect two given vertex elements. These methods usually need to be overridden in self-defined model elements.

Figure 4 gives a rough impression of the generic metamodel. Some of the important classes are depicted in the upper section. The lower section shows how (part of) a metamodel for UML state machines could be derived from the generic classes. Having an obviously vertex-like characteristic, UmlState and UmlCompositeState would be derived from ModelVertex. UmlTransition would become a derivative of ModelEdge. All three classes override syntax-enforcing methods as necessary to ensure the rules of the UML specification: states may be connected using transitions, and composite states may contain other states.

The same user interface works on all application-specific metamodels without adjustments, since the basic metamodel elements provide introspection mechanisms that allow querying an element's attributes and associations with other elements, and potentially also allow their modification. These mechanisms are used for loading, saving, and printing models on a generic basis as well as for providing a foundation on which the more sophisticated user interface components mentioned above can be built.

5 Experiences

Our framework for lightweight object-oriented tools is the result of an iterative process. First the software architecture editor was developed from scratch without any framework in mind. Then the need for a tool to tailor the Unified Process arose and further lightweight tools for object-oriented design techniques followed. While developing the other tools, we identified several features common to the different tools and decided to evolve a framework that allows efficiently developing lightweight object-oriented design tools. Through several refactoring steps we separated the generic functionality from the tool-specific ones and achieved the framework and metamodel presented in sections 3 and 4.

Our experiences during development of different lightweight tools for object-oriented design techniques using our framework (see section 2) showed that the framework is worth the effort we spent on evolving it. Instead of repeatedly starting from scratch, the framework and metamodel allows efficient development of lightweight tools. There are well-defined steps, using the hooks provided by the framework, the developer has to follow to achieve a running application:

1. build a data model for the tool through inheriting the framework's metamodel
2. develop graphical representations of the data model's elements through inheriting the framework's generic figures
3. customize the layout and functionality adapting the generic components provided by the framework (if necessary)
4. integrate additional functionality (such as a simulation component)

We experienced that the framework is a stable and powerful base suitable for different tools devoted to teaching and learning various aspects of object-oriented development.

Currently, we are releasing a first stable version of the tools. They will be used in the lecture on software engineering during this spring/summer term for the first time and their impact will be evaluated. We hope that the clear focus of the lightweight tools will gain a positive acceptance among students. We assume that the tools' short startup times and low memory foot print are further benefits.

6 Outlook

Our next step is to extend the above-mentioned simulation engine of the statechart editor by a multimedia interface. This will allow incorporation of arbitrary media elements into the simulation, thereby simulating or "controlling" real world applications or devices by the statemachine modeled. Different media elements might be used to illustrate the application in its different states. A typical example would be a simulation of Harel's watch [Har87], illustrating the state changes using pictures of a digital watch. Using the multimedia component we hope to bridge the gap between abstract formal models and real world applications. We assume that is possible to use the same approach to illustrate the dynamics of software architectures as well.

In parallel to the extended simulation engine, we develop a project tutor for applying the Unified Process. Based on the tailored model instantiated by the process modeler, the tutor provides additional support for the students during execution of the project's development process (e.g. gives hints what

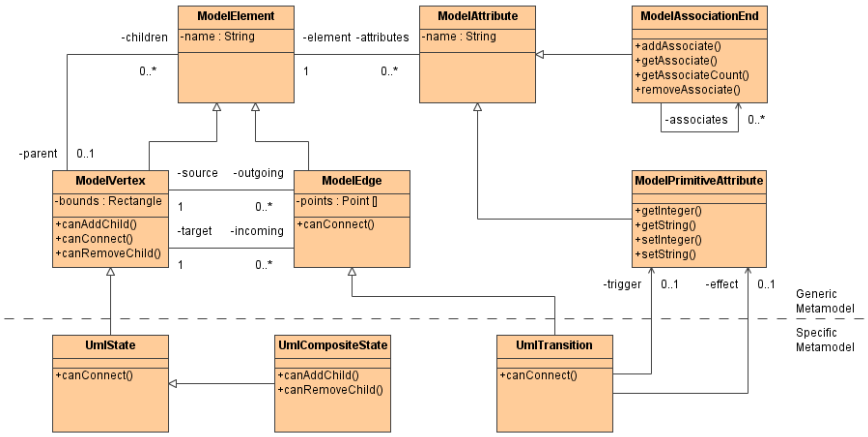


Figure 4: Simplified generic and specific metamodel

to do next). The tutor is also based on our framework illustrating the broad range of tools which can be built onto the framework.

To complete the set of our tools, a class diagram editor is the obvious choice. Due to the metamodel approach of our framework, such an editor would a suitable starting point for studying model driven architecture approaches [Fra03]. In this case, the tool and its internals would become part of the lecture. In the ideal case, it would also allow us developing the metamodels for further lightweight tools with the tool itself.

References

[DE02] Ernst-Erich Doberkat and Gregor Engels. MuSofT – Multimedia in der SoftwareTechnik. *Informatik Forschung und Entwicklung*, 17(1), January 2002.

[Fra03] David S. Frankel. *Model Driven Architecture – Applying MDA to Enterprise Computing*. OMG Press, 2003.

[GHJV97] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1997.

[Har87] David Harel. Statecharts: A Visual Formulation for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[HN96] David Harel and Amnon Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[JBR98] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1998.

[Kai01] Wolfram Kaiser. Become a programming Picasso with JHotDraw, 2001. `http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-jhotdraw.html`.

[Kel02] Udo Kelter. Gestaltungsrichtlinien für Editor-Simulatoren für graphartige Dokumente. In Sigrid Schubert, Bernd Reusch, and Norbert Jesse, editors, *Informatik bewegt. 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 30. Sept.–3. Okt. 2002 in Dortmund*, pages 393–400. Gesellschaft für Informatik, 2002.

[OMG02] Object Management Group. *Meta Object Facility (MOF) Specification 1.5*, 2002. `http://www.omg.org/cgi-bin/doc?formal/2002-04-03`.

[OMG03] Object Management Group. *Unified Modeling Language (UML) 1.5 Specification*, 2003. `http://www.omg.org/cgi-bin/doc?formal/03-03-01`.

[SGW94] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley and Sons, 1994.

[SR98] Bran Selic and Jim Rumbaugh. Using UML for Modeling Complex Real-Time Systems. 1998. `http://www.rational.com/products/whitepapers/442.jsp`.

/99/ T. Bühren, M. Cakir, E. Can, A. Dombrowski, G. Geist, V. Gruhn, M. Gürgrn, S. Handschumacher, M. Heller, C. Lüer, D. Peters, G. Vollmer, U. Wellen, J. von Werne
Endbericht der Projektgruppe eCCo (PG 315)
Electronic Commerce in der Versicherungsbranche
Beispielhafte Unterstützung verteilter Geschäftsprozesse
Februar 1999

/100/ A. Fronk, J. Pleumann,
Der DoDL-Compiler
August 1999

/101/ K. Alfert, E.-E. Doberkat, C. Kopka
Towards Constructing a Flexible Multimedia Environment for Teaching the History of Art
September 1999

/102/ E.-E. Doberkat
An Note on a Categorial Semantics for ER-Models
November 1999

/103/ Christoph Begall, Matthias Dorka, Adil Kassabi, Wilhelm Leibel, Sebastian Linz, Sascha Lüdecke, Andreas Schröder, Jens Schröder, Sebastian Schütte, Thomas Sparenberg, Christian Stücke, Martin Uebing, Klaus Alfert, Alexander Fronk, Ernst-Erich Doberkat
Abschlußbericht der Projektgruppe PG-HEU (326)
Oktober 1999

/104/ Corina Kopka
Ein Vorgehensmodell für die Entwicklung multimedialer Lernsysteme
März 2000

/105/ Stefan Austen, Wahid Bashirazad, Matthais Book, Traugott Dittmann, Bernhard Flechtker, Hassan Ghane, Stefan Göbel, Chris Haase, Christian Leifkes, Martin Mocker, Stefan Puls, Carsten Seidel, Volker Gruhn, Lothar Schöpe, Ursula Wellen
Zwischenbericht der Projektgruppe IPSI
April 2000

/106/ Ernst-Erich Doberkat
Die Hofzwerge — Ein kurzes Tutorium zur objektorientierten Modellierung
September 2000

/107/ Leonid Abelev, Carsten Brockmann, Pedro Calado, Michael Damatow, Michael Heinrichs, Oliver Kowalke, Daniel Link, Holger Lümkemann, Thorsten Niedzwetzki, Martin Otten, Michael Rittinghaus, Gerrit Rothmaier
Volker Gruhn, Ursula Wellen
Zwischenbericht der Projektgruppe Palermo
November 2000

/108/ Stefan Austen, Wahid Bashirazad, Matthais Book, Traugott Dittmann, Bernhard Flechtker, Hassan Ghane, Stefan Göbel, Chris Haase, Christian Leifkes, Martin Mocker, Stefan Puls, Carsten Seidel, Volker Gruhn, Lothar Schöpe, Ursula Wellen
Endbericht der Projektgruppe IPSI
Februar 2001

/109/ Leonid Abelev, Carsten Brockmann, Pedro Calado, Michael Damatow, Michael Heinrichs, Oliver Kowalke, Daniel Link, Holger Lümkemann, Thorsten Niedzwetzki, Martin Otten, Michael Rittinghaus, Gerrit Rothmaier
Volker Gruhn, Ursula Wellen
Zwischenbericht der Projektgruppe Palermo
Februar 2001

/110/ Eugenio G. Omodeo, Ernst-Erich Doberkat
Algebraic semantics of ER-models from the standpoint of map calculus.
Part I: Static view
März 2001

/111/ Ernst-Erich Doberkat
An Architecture for a System of Mobile Agents
März 2001

/112/ Corina Kopka, Ursula Wellen
Development of a Software Production Process Model for Multimedia CAL Systems by Applying Process Landscaping
April 2001

/113/ Ernst-Erich Doberkat
The Converse of a Probabilistic Relation
Oktober 2002

/114/ Ernst-Erich Doberkat, Eugenio G. Omodeo
Algebraic semantics of ER-models in the context of the calculus of relations.
Part II: Dynamic view
Juli 2001

/115/ Volker Gruhn, Lothar Schöpe (Eds.)
Unterstützung von verteilten Softwareentwicklungsprozessen durch integrierte Planungs-, Workflow- und Groupware-Ansätze
September 2001

/116/ Ernst-Erich Doberkat
The Demonic Product of Probabilistic Relations
September 2001

/117/ Klaus Alfert, Alexander Fronk, Frank Engelen
Experiences in 3-Dimensional Visualization of Java Class Relations
September 2001

/118/ Ernst-Erich Doberkat
The Hierarchical Refinement of Probabilistic Relations
November 2001

/119/ Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer, Ingo Röpling,
Clemens Schäfer, Nikolai Schreier, Olga Shtern
Ursula Wellen, Dirk Peters, Volker Gruhn
Project Group Chairware Intermediate Report
November 2001

/120/ Volker Gruhn, Ursula Wellen
Autonomies in a Software Process Landscape
Januar 2002

/121/ Ernst-Erich Doberkat, Gregor Engels (Hrsg.)
Ergebnisbericht des Jahres 2001
des Projektes “MuSofT – Multimedia in der SoftwareTechnik”
Februrar 2002

/122/ Ernst-Erich Doberkat, Gregor Engels, Jan Hendrik Hausmann, Mark Lohmann, Christof Veltmann
Anforderungen an eine eLearning-Plattform — Innovation und Integration —
April 2002

/123/ Ernst-Erich Doberkat
Pipes and Filters: Modelling a Software Architecture Through Relations
Juni 2002

/124/ Volker Gruhn, Lothar Schöpe
Integration von Legacy-Systemen mit Eletronic Commerce Anwendungen
Juni 2002

/125/ Ernst-Erich Doberkat
A Remark on A. Edalat’s Paper *Semi-Pullbacks and Bisimulations in Categories of Markov-Processes*
Juli 2002

/126/ Alexander Fronk
Towards the algebraic analysis of hyperlink structures
August 2002

/127/ Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer
Ingo Röpling, Clemens Schäfer, Nikolai Schreier, Olga Shtern
Ursula Wellen, Dirk Peters, Volker Gruhn
Project Group Chairware Final Report
August 2002

/128/ Timo Albert, Zahir Amiri, Dino Hasanbegovic, Narcisse Kemogne Kamdem,
Christian Kotthoff, Dennis Müller, Matthias Niggemeier, Andre Pavlenko, Stefan Pinschke,
Alireza Salemi, Bastian Schlich, Alexander Schmitz
Volker Gruhn, Lothar Schöpe, Ursula Wellen
Zwischenbericht der Projektgruppe Com42Bill (PG 411)
September 2002

/129/ Alexander Fronk
An Approach to Algebraic Semantics of Object-Oriented Languages
Oktober 2002

/130/ Ernst-Erich Doberkat
Semi-Pullbacks and Bisimulations in Categories of Stochastic Relations
November 2002

/131 Yalda Ariana, Oliver Effner, Marcel Gleis, Martin Krzysiak,
Jens Lauert, Thomas Louis, Carsten Röttgers, Kai Schwaighofer,
Martin Testrot, Uwe Ulrich, Xingguang Yuan
Prof. Dr. Volker Gruhn, Sami Beydeda
Endbericht der PG nightshift:
Dokumentation der verteilten Geschäftsprozesse im FBI und Umsetzung von Teilen dieser Prozesse im Rahmen eines FBI-Intranets basierend auf WAP- und Java-Technologie
Februar 2003

/132/ Ernst-Erich Doberkat, Eugenio G. Omodeo
ER Modelling from First Relational Principles
Februar 2003

/133/ Klaus Alfert, Ernst-Erich Doberkat, Gregor Engels (Hrsg.)
Ergebnisbericht des Jahres 2002 des Projektes "MuSofT – Multimedia in der SoftwareTechnik"
März 2003

/134/ Ernst-Erich Doberkat
Tracing Relations Probabilistically
März 2003

/135/ Timo Albert, Zahir Amiri, Dino Hasanbegovic, Narcisse Kemogne Kamdem,
Christian Kotthoff, Dennis Müller, Matthias Niggemeier,
Andre Pavlenko, Alireza Salemi,Bastian Schlich, Alexander Schmitz,
Volker Gruhn, Lothar Schöpe, Ursula Wellen
Endbericht der Projektgruppe Com42Bill (PG 411)
März 2003

/136/ Klaus Alfert
Vitruv: Specifying Temporal Aspects of Multimedia Presentations —
A Transformational Approach based on Intervals
April 2003

/137/ Klaus Alfert, Jörg Pleumann, Jens Schröder
A Framework for Lightweight Object-Oriented Design Tools
April 2003