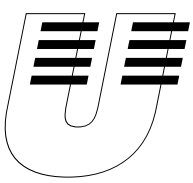

UNIVERSITÄT
KOBLENZ · LANDAU

Meta-CASE Worldwide

Jürgen Ebert, Roger Süttenbach,
Ingar Uhe

24/98

Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Rheinau 1, D-56075 Koblenz
E-mail: researchreports@infko.uni-koblenz.de,
WWW: http://www.uni-koblenz.de/fb4/

Meta-CASE Worldwide

Jürgen Ebert, Roger Süttenbach, Ingar Uhe
University of Koblenz-Landau – Institute for Software Technology

November 30, 1998

Abstract

The huge success of the Internet encourages existing systems to evolve by using this technology. This paper presents the meta-CASE system JKogge which is designed like a CASE browser and integrates CASE and Internet technology.

Keywords: Meta-CASE, CASE, declarative modeling, software engineering environments, Internet, distributed graphs, VRML97.

1 Introduction

Internet
The overwhelming success of the Internet – especially the World Wide Web (WWW) – has its effect on all fields of software engineering. Not only technical requirements and problems resulting from highly distributed documents and systems but also changes of user expectations have to be considered.

One main expectation or idea seems to crystallize: 'all is in the net'. That means the work space of a user is the Internet. In this scenario the users get their resources – systems and documents – through the Internet. It is irrelevant if a resource has its location in Chicago, Berlin, or Koblenz, because only the addresses differ, nothing else. And everything should happen the easy 'point and click' way of the WWW.

Component-Oriented Development
The success of the Internet and therefore the need for distributed resources also gives software component technology a push. This technology is closely related to 'distributed scenarios' which perfectly describe the situation in the Internet. Although the request for component technology is very old [M68],

although the idea of modularization is well-known and established (e.g. in Eiffel, ADA, Oberon) and although the existence of component systems (e.g. DCOM, CORBA, JavaBeans) is reality, there is still a lack of a clear definition of components. Many definitions, for example in Booch [B87] or Jacobson [J93], only demand the property of modularity. Industrial standards like DCOM declare anything a component which implements their interfaces. Others like Orfali et al. [OHE96] and Sametinger [Sa97] add the request for a defined environment and standard interfaces for interoperability.

A very restrictive definition is made by Szyperski [S97] based on the results of an ECOOP workshop. He states that a component is a unit which can be deployed independently, which is used by third-parties, and which has no persistent state.

We refer – with limitations – to this definition. We will precise our use of components in Section 2.3.

CASE

The demands emerging from the use of the Internet as well as the use of components have strong influence on the production and use of CASE and meta-CASE tools.

Traditional meta-CASE tools and the CASE tools, which are built by them (e.g. [ESU96], [DK95], [KLR96], [JJQ98]), are more or less large monolithic systems. These systems follow a client server approach but their focus is on processing local documents. The user interface is also a traditional one.

But with the Internet in mind the requirements of the users lead to CASE tools which act and look like *CASE browsers*. Figure 1 sketches this concept.

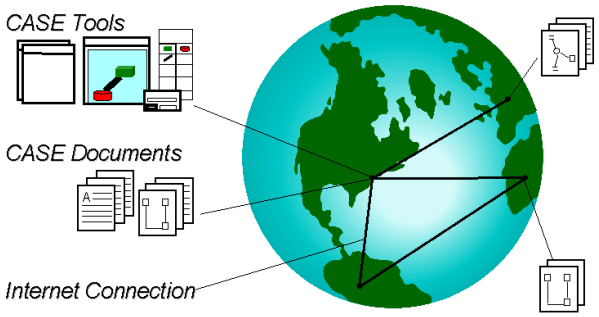

Figure 1: CASE Tools and Documents in the Internet Context

The scenario would be as follows: The user clicks on a certain link on a WWW page, which represents a CASE document, and the needed CASE tool for this document is transferred to her or his local machine from any

place in the intra- or Internet. All functionality needed is loaded from the intra- or Internet to the workstation of the user.

In this paper we describe how CASE and Internet technology concepts were integrated. The main ideas of the implemented system – called *JKogge* – are pointed out in **Section 2**. An example for components in JKogge is shown in **Section 3**. **Section 4** finally describes the current state of the JKogge system and gives an outlook to its further development.

2 JKogge

JKogge realizes the integration of CASE and Internet concepts sketched in the introduction. The three main new elements of JKogge are

- the concept of *distributed graphs*,
- the idea of processing graphs by *components* which are realized by plug-ins,
- a *base system* which is fairly small and only responsible for loading documents and components.

JKogge is based on the meta-CASE system KOGGE [ESU96] and its main ideas which are presented in the next section. This description is followed by an explanation of the three concepts mentioned above. The chapter is completed by a short presentation of the implementation.

2.1 KOGGE

JKogge is based on KOGGE [ESU96], a meta-CASE system which allows the generation of tools for visual languages. The system has been used to produce editors for visual languages used in software engineering – like data flow diagrams [D96], entity relationship diagrams, class diagrams [KU96] – as well as for languages used to specify services in the telecommunication business [P98], and for software evaluation [PL97].

The basic idea of KOGGE is to specify tools and not to program every single one of them. In order to achieve this there is a separation between a *base system* common for all tools and a *tool description* which is unique for each tool (see Figure 2). A specific KOGGE tool results from the interpretation of the tool description by the base system.

The separation between tool description and base system is invisible to the users of KOGGE tools, both parts act together like a single system.

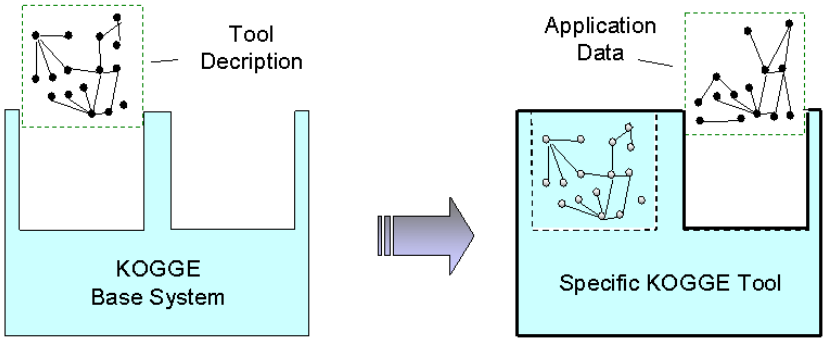

Figure 2: Structure of KOGGE Tools

The tool description contains the aspects specific to a KOGGE tool for a certain visual language and is divided into three parts describing: the concepts of the visual language, the structure of the menus and the interaction with the user. There is a distinct description formalism for all these three parts: extended entity relationship diagrams augmented by a constraint language (EER/GRAL)[EF94], menu graphs and state charts.

The core functionality which is used for all tools is provided by the base system which is programmed in C++ and uses the CAD-system VARIOCAD [DRB93],[Me95] for visualization.

As mentioned above, KOGGE was successfully used to build a variety of tools. The aim of the JKogge system is on the one hand to provide a framework which preserves the main concepts of KOGGE (like the separation of tool description and base functionality) but on the other hand to incorporate the handling of tools used in the Internet context. In addition to this integration a number of further changes were also made. The functionality of the base system was greatly reduced and the graphical modules are not realized by libraries any more but by plug-ins which achieve this functionality in a more declarative manner.

2.2 Distributed Graphs

KOGGE already used graphs in order to store the tool description and the data produced by the application (see Figure 2). Since JKogge deals with a distributed scenario, the concept of the graphs used was modified. In JKogge a *distributed graph* [BS98] is a graph which is decomposed into a number of physical graphs (subgraphs). These subgraphs, related by shared vertices and connecting edges, are called *documents*.

This definition implies that each subgraph can be loaded, processed, and

stored independently. This is achieved by mapping the coherent logical view of graphs to a distributed physical structure. In the logical view, illustrated in Figure 3, there are elements which belong to several graphs (like *vertex* v and *edge* e).

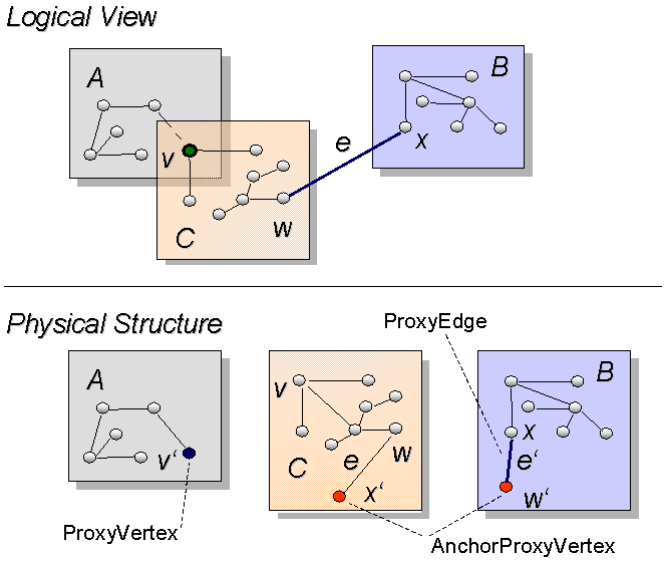

Figure 3: Logical and Physical View in JKogge

In the physical structure these common elements are represented by *proxy vertices* and *proxy edges*. I.e. there are vertices and edges which have a special type – ProxyVertex, AnchorProxyVertex for vertices and ProxyEdge for edges – referring to vertices or edges in other graphs. In JKogge graphs are identified by their locations which are described as Internet addresses (following the URL convention).

2.3 Components in JKogge

Components are realized by *JKogge-plug-ins*. The use and the meaning of plug-in in this context is quite similar to the use of this term in other software systems, like in Eudora^{TM} or Netscape^{TM}. The latter, for example, defines a plug-in as: ".. software programs that extend the capabilities of Netscape in a specific way". Following this definition a JKogge-plug-in extends the capabilities of the JKogge base system. From a different point of view JKogge-plug-ins can be seen as components which are built for specific document types. A JKogge-plug-in does *process* documents of a certain document type. In principle, this *processing* is arbitrary and up to the component in question. Thus, JKogge-plug-ins meet Szyperski's definition (see

p. 2) with the additional demand that there has to be a base system loading the plug-ins. Typical examples of JKogge-plug-ins[1] are viewers, editors or interpreters of documents.

Structure
In principle a plug-in is a unit with its own thread of control which is able to process documents. Figure 4 gives a schematic description of a plug-in.

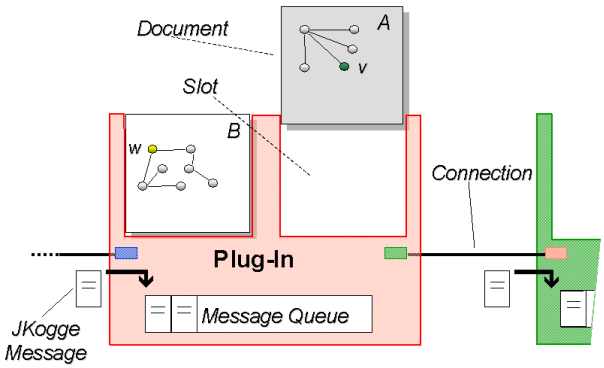

Figure 4: Structure and Communication of JKogge-Plug-Ins

Plug-ins are designed to process documents. The number and the type of documents, which a plug-in can process, is defined by its slots. A *slot* is a named and typed field in which a document can be put in. The type of the slot describes the type of the document which can be contained. Documents can be added to or removed from the slots at run-time.

Communication
The communication among the plug-ins in the base system is an asynchronous message-passing. I.e. there are message objects – called *JKogge-messages* – which are exchanged between plug-ins. Some messages are requests issued to one plug-in by another, other messages deliver data or notifications.

A JKogge-message is an object sent from one plug-in to another. It contains information about the sender, the specific command to be executed, a document and specific vertices in the document. The messages are processed in the order in which they are received. At one point of time a plug-in can only process one message. Because each plug-in has its own thread of control, other messages can be put in a *message queue* – concurrently – by other plug-ins but processing is delayed until the processing of the message worked on is finished.

[1] In the following we will use the term plug-in instead of JKogge-plug-in.

2.4 Base System

The base system is responsible for loading plug-ins and documents. It supports not only access to the local file system but also supports access via Internet addresses. Figure 5 shows a schematic description of the base system. It can be seen that a number of connected documents and three plug-ins have been loaded.

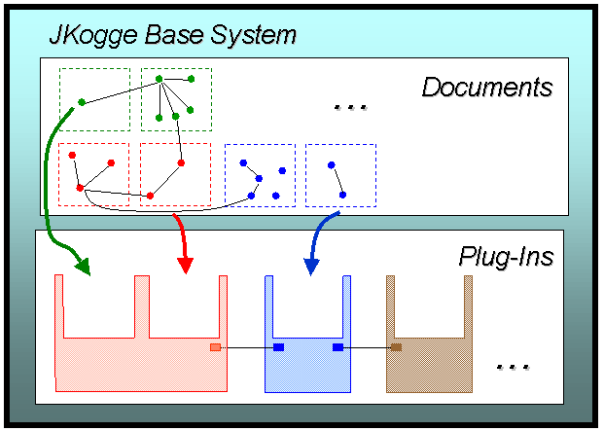

Figure 5: Base System

Connected documents are loaded by the base system on demand. I.e. only if a plug-in explicitly requests a document, this document will be loaded.

The second major task of the base system is to serve as a *mediator* (see [GHJ+95] between the plug-ins. When a plug-in is loaded it automatically gets connected to the base system. Thus, the plug-in is able to get a connection to another plug-in by the base system without knowing specific details about its location. The connection between different plug-ins is used to exchange JKogge-messages.

Further tasks of the base system are the management of user information, including authorization, and providing services for customizing the user's environment.

2.5 Implementation

The JKogge base system and the plug-ins are implemented in Java. The graphs used throughout the system are stored and manipulated with a derivative (*JGraLab*) of a software called *GraLab* [DW98].

All plug-ins work in their own thread of control. They all share a common interface and are packed into Java archive files using some of the bean

features. The base system is able to load these `.jar` files[2] and unpack them. The implementation was done by applying design patterns [GHJ+95] extensively. The JGraLab for example employs the abstract factory pattern to choose whether graphs are stored in an ObjectStore^{TM} database or as ordinary files.

3 Components in JKogge

As described in the last section, a JKogge tool consists of a small base system and a number of plug-ins which fulfill small, limited tasks. Following the KOGGE approach, a CASE tool realized with JKogge consists of a set of tool descriptions, the base system and a number of plug-ins. These plug-ins do for example interpret the tool description like a state chart interpreter, an action interpreter or e.g. take care of the visualization of the user interface.

This section describes some of the plug-ins, how they work and how they communicate with each other. The plug-ins presented take care of the visualization of the windows and dialogs used in a concrete tool.

3.1 Plug-ins

A dialog used in a CASE tool could look like this:

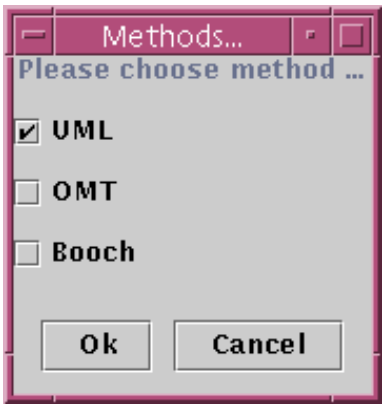

Figure 6: Dialog Used in a CASE Tool

There are JKogge plug-ins which allow to specify the windows and dialogs

[2] To illustrate the size of the files some numbers:
– base system: appr. 70 classes/interfaces and 90 kbyte
– plug-ins: appr. 2-25 classes and 5-70 kbyte
– JGraLab: 100 classes/interfaces and 80 kbyte

needed. This task is realized by three plug-ins: a *parser-*, a *create-*, and a *visualize plug-in*.

At present the dialogs needed are specified by a simple textual language. The description is parsed into a graph which serves as a template for the windows and dialogs which can be used in the tool. In order to actually use these dialogs a working copy of the template graph has to be made which is the task of the create plug-in. The visualize plug-in takes care of the visualization and updating of the windows used. Figure 7 shows the cooperation of the plug-ins.

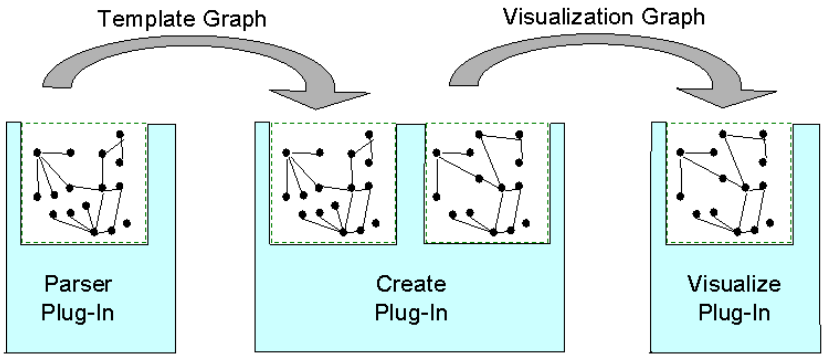

Figure 7: Plug-ins for the Visualization of Dialogs and Windows

Graphs are used as internal data structures because this integrates all parts of the JKogge-tool description into a homogeneous format. They offer an efficient way to store and retrieve information and they are – as described in section 2.2 – capable of the realization of distributed documents.

Parser Plug-In

As stated above windows are described by a textual language. The lines below are part of the description of the dialog shown in Figure 6.

```
FRAME Frame ( title="Methods...",
resizable=yes,
isEnabled=yes,
contentIsPlacedBy Border );

LABEL Label ( title="Please choose method ... ",
alignment=center,
isChildOf Frame,
isEnabled=yes,
placedByBorderLayout Border ( position=North ) );

FLOWLAYOUT Flow ( alignment=center,
...
```

As described above the result of the parsing process is a graph of dialog templates.

Create Plug-In

These template graphs are instantiated by the create plug-in which means that it generates a second graph which can be customized and then be visualized. An example for such a customizing process could be to fill a text field with a certain string before the corresponding window is visualized. There is a distinction between the template and the visualization graph to make it possible to have one template for a dialog and various instances of it at the same time which only differ in the labels of the buttons for example.

Visualize Plug-In

The concrete dialogs are described in the visualization graph. This visualization graph can be visualized by the visualize plug-in. Furthermore the visualize plug-in can hide and destroy dialogs and windows and update the visualization according to changed values in the visualization graph.

3.2 Communication

As described above some plug-ins (like the visualize plug-in) can fulfill various task. This implies that there has to be a means of communication between plug-ins. They communicate via JKogge-messages as described in Section 2.3. The visualize plug-in accepts the commands (which are part of the JKogge-message): *show, hide, destroy* and *update*. But what is even more important is the communication back from dialogs and windows visualized on screen to the visualize plug-in. The visualize plug-in works as listener for the events emerging which means that it receives the events from the Java runtime system. It passes them on to the other plug-ins of the system. For example, if the user clicks the `Ok-button` the rest of the system has to be notified to be able to react to the user interaction. In JKogge the specifier of a tool interface specifies which events can be delivered like in the following example:

```
BUTTON OKButton ( title="Ok",
isEnabled=yes,
isChildOf ButtonPanel,
triggers {(ACTION_PERFORMED,"Ok-Selected")},
placedByFlowLayout Flow);
```

In this example the visualize-plug in creates a JKogge-message which tells the component which uses the dialog that the specified `Ok-Selected` event occurred. The reaction to this is up to the plug-in which receives the JKogge-message.

4 Outlook

The previous sections showed that a great variety of tools can be realized with JKogge's component-oriented approach. The JKogge project was inspired and was able to profit from the experience gained in building CASE tools with the KOGGE meta-CASE system. JKogge is a successor of KOGGE with new added features.

The base system successfully works with the plug-ins realized so far. Only some of the existing plug-ins were described, some are already implemented, and some will be implemented shortly.

New plug-ins for various contexts will be designed, too. This could include visual editors, parsers, analyzers and simulators, graph browsers, viewers for any kind of text, HTML text generators, and a GUI builder to replace the textual description of dialogs and windows.

In CASE tools a wide variety of diagrams can be employed. Plug-ins used for the visualization could be used to drive a visualization in a VRML [V97], [LMM96] browser for example. Another alternative examined is the realization of such a visualization facility in Java3D [S98]. Because of JKogge's architecture based on components both can be used interchangeably.

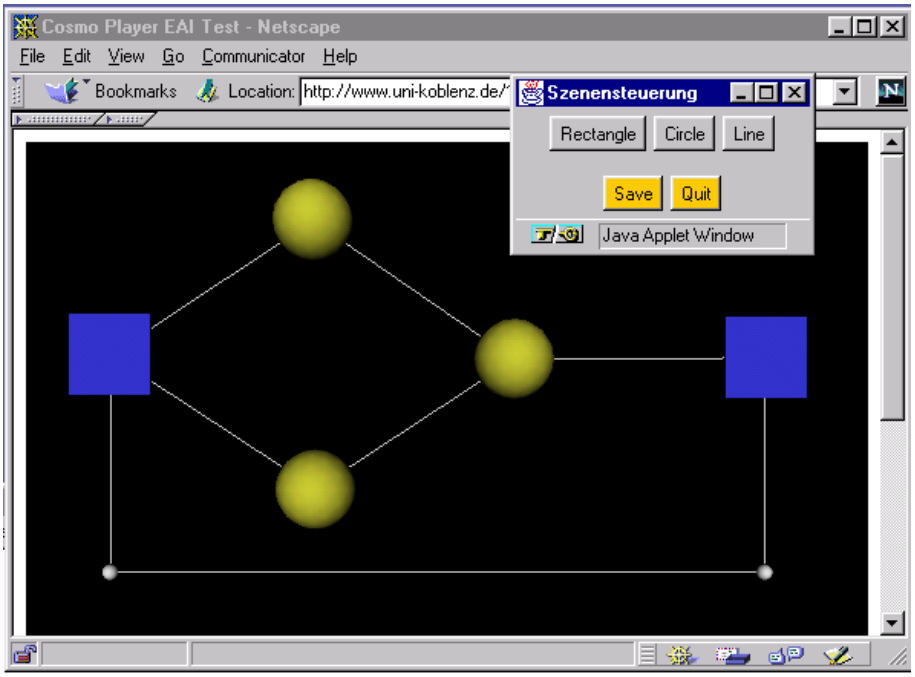

Figure 8: Diagram Visualization using VRML

A plug-in using VRML for the visualization of diagrams could look like

the screenshot shown in Figure 8.

All in all one can say it is already possible to build tools with JKogge. The concept of a small base system which loads plug-ins which can be built to provide almost any functionality needed gives the system the flexibility to realize tools for a wide variety of purposes.

Acknowledgement: The authors thank Martin Schulze, Prof. Dr. Manfred Rosendahl, and all the students working on the projects KOGGE and JKogge.

References

[B87] Booch, G.; *Software Components with Ada: Structures, Tools, and Subsystems.* Redwood City: Cummings, 1994[2].

[DK95] Däberitz, D.; Kelter, U.; *Rapid Prototyping of Graphical Editors in an Open SDE.* In: Proc. 7th Conf. on Software Engineering Environments (SEE '95), p. 61-72. Noordwijkerhout: IEEE Computer Society Press, 1995.

[DW98] Dahm, P.; Widmann, F.; *Das Graphenlabor.* Koblenz: Universität Koblenz-Landau, Fachberichte Informatik 12/98, 1998.

[BS98] Behling, M.; Süttenbach, R.; *Verteilte Graphen in der JKogge.* Koblenz: Universität Koblenz-Landau, Projektbericht, 1998.

[D96] Drüke, M.; *Dokumentation für den Datenflu̇sdiagramm-Editor.* Koblenz: Universität Koblenz-Landau, Studienarbeit, 1996.

[DRB93] Du, C.; Rosendahl, M.; Berling, R.; *Variation of Geometry and Parametric Design.* In: Proc. 3rd. International Conference on CAD and Computer Graphics, Beijing, Aug. 23-26, 1993, p. 400-405, International Academic Publishers, 1993.

[EWD+96] Ebert, J.; Winter, A.; Dahm, P.; Franzke, A.; Süttenbach, R.; *Graph Based Modeling and Implementation with EER/GRAL.* In: B. Thalheim [Ed.]; 15th International Conference on Conceptual Modeling (ER'96), Lecture Notes in Computer Science, 1157. Berlin: Springer, 1996.

[ESU96] Ebert, J.; Süttenbach, R.; Uhe, I.; *Meta-CASE in Practice: A Case for KOGGE.* In: A. Olivé and J.A. Pastor [Eds.]; 9th International Conference, CAiSE '96, Lecture Notes In Computer Science, 1250. Berlin: Springer, 1997.

[EF94] Ebert, J.; Franzke, A.; *A Declarative Approach to Graph Based Modeling.* In: Mayr, E.; Schmidt, G.; Tinhofer, G. [Eds.]; Graphtheoretic Concepts

in Computer Science, Lecture Notes in Computer Science, 903, p. 38-50. Berlin: Springer, 1995.

[F96] Franzke, A.; *GRAL: A Reference Manual.* Koblenz: Universität Koblenz-Landau, Fachberichte Informatik 11/96, 1996.

[GHJ+95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; *Design Patterns: Elements of Reusable Object-oriented Software.* Reading: Addison-Wesley, 1995.

[G98] Griffel, F.; *Componentware - Konzepte und Techniken eines Softwareparadiagmas.* Heidelberg: dpunkt.verlag, 1998.

[LMM96] Lea, R.; Matsuda, K; Miyashita, K. *Java for 3D and VRML Worlds.* Indianapolis: New Riders , 1996.

[J93] Jacobson, I.; *Object-Oriented Software Engineering.* Reading: Addison-Wesley, 1993.

[JJQ98] Jarke, M.; Jeusfeld, M.A.; Quix, C. [Eds.]; *ConceptBase V5.0 User Manual.* RWTH Aachen, 1998 (`http://www-i5.informatik.rwth-aachen.de/CBdoc/userManual-V50/`).

[KLR96] Kelly, S.; Lyytinen, K.; Rossi, M.; *MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment.* In: Constantopoulos, P.; Mylopoulos, J.; Vassiliou, Y. [Eds.]; Advanced Information System Engineering, Lecture Notes in Computer Science, 1080. Berlin: Springer, 1996.

[KU96] Kölzer, A.; Uhe, I.; *Benutzerhandbuch für das KOGGE-Tool BONsai III.* Koblenz: Universität Koblenz-Landau, Projektbericht, 1996.

[M68] McIlroy, M.D.; *Mass Produced Software Components.* In: Naur. P and Randell, B. [Eds.]; NATO Conference on Software Engineering, Proccedings. Brüssel: NATO Science Committee, 1968.

[Me95] Meissner, A.; *GRABE – Eine objekt-orientierte Sprache zur Spezifikation von Symbolen in interaktiven graphischen Editoren.* Koblenz: Fölbach, Dissertation, 1995.

[OHE96] Orfali, R.; Harkey, D; Edwards, J.; *The Essential Distributed Objects Survival Guide.* New York: Wiley & Sons, 1996.

[P98] Polock, D.; *Benutzerhandbuch für die KOGGE-Instanz FAKT.* Koblenz: Universität Koblenz-Landau, Projektbericht, 1998.

[PL97] Pühler, T.; Löcher, M.: *Entwicklung eines Softwareevaluationstools* Koblenz: Universität Koblenz-Landau, Studienarbeit, 1997.

[V97] (`http: http://www.vrml.org/Specifications/VRML97/`)

[Sa97] Sametinger, J.; *Software Engineering with Reusable Components.* Berlin: Springer, 1997.

[S98] Sun Microsystems; *Java 3D API Specification.* Version 1.1 Beta 02, 1998 (`http://java.sun.com/products/java-media/3D/forDevelopers/j3dguide/j3dTOC.doc.html`)

[S97] Szyperski, C.; *Component Software – Beyond Object-Oriented Programming.* Reading: Addison-Wesley, 1997.

Available Research Reports (since 1995):

1998

24/98 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE Worldwide.

23/98 *Peter Baumgartner, Norbert Eisinger, Ulrich Furbach.* A Confluent Connection Calculus.

22/98 *Bernt Kullbach, Andreas Winter.* Querying as an Enabling Technology in Software Reengineering.

21/98 *Jürgen Dix, V.S. Subrahmanian, George Pick.* Meta-Agent Programs.

20/98 *Jürgen Dix, Ulrich Furbach, Ilkka Niemelä .* Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations.

19/98 *Jürgen Dix, Steffen Hölldobler.* Inference Mechanisms in Knowledge-Based Systems: Theory and Applications (Proceedings of WS at KI '98).

17/98 *Stefan Brass, Jürgen Dix, Teodor C. Przymusinski.* Super Logic Programs.

16/98 *Jürgen Dix.* The Logic Programming Paradigm.

15/98 *Stefan Brass, Jürgen Dix, Burkhard Freitag, Ulrich Zukowski.* Transformation-Based Bottom-Up Computation of the Well-Founded Model.

14/98 *Manfred Kamp.* GReQL – Eine Anfragesprache für das GUPRO–Repository – Sprachbeschreibung (Version 1.2).

12/98 *Peter Dahm, Jürgen Ebert, Angelika Franzke, Manfred Kamp, Andreas Winter.* TGraphen und EER-Schemata – formale Grundlagen.

11/98 *Peter Dahm, Friedbert Widmann.* Das Graphenlabor.

10/98 *Jörg Jooss, Thomas Marx.* Workflow Modeling according to WfMC.

9/98 *Dieter Zöbel.* Schedulability criteria for age constraint processes in hard real-time systems.

8/98 *Wenjin Lu, Ulrich Furbach.* Disjunctive logic program = Horn Program + Control program.

7/98 *Andreas Schmid.* Solution for the counting to infinity problem of distance vector routing.

6/98 *Ulrich Furbach, Michael Kühn, Frieder Stolzenburg.* Model-Guided Proof Debugging.

5/98 *Peter Baumgartner, Dorothea Schäfer.* Model Elimination with Simplification and its Application to Software Verification.

4/98 *Bernt Kullbach, Andreas Winter, Peter Dahm, Jürgen Ebert.* Program Comprehension in Multi-Language Systems.

3/98 *Jürgen Dix, Jorge Lobo.* Logic Programming and Nonmonotonic Reasoning.

2/98 *Hans-Michael Hanisch, Kurt Lautenbach, Carlo Simon, Jan Thieme.* Zeitstempelnetze in technischen Anwendungen.

1/98 *Manfred Kamp.* Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools — A Generic Approach.

1997

32/97 *Peter Baumgartner.* Hyper Tableaux — The Next Generation.

31/97 *Jens Woch.* A component-based and abstractivistic Agent Architecture for the modelling of MAS in the Social Sciences.

30/97 *Marcel Bresink.* A Software Test-Bed for Global Illumination Research.

29/97 *Marcel Bresink.* Deutschsprachige Terminologie des Radiosity-Verfahrens.

28/97 *Jürgen Ebert, Bernt Kullbach, Andreas Panse.* The Extract-Transform-Rewrite Cycle - A Step towards MetaCARE.

27/97 *Jose Arrazola, Jürgen Dix, Mauricio Osorio.* Confluent Rewriting Systems for Logic Programming Semantics.

26/97 *Lutz Priese.* A Note on Nondeterministic Reversible Computations.

25/97 *Stephan Philippi.* System modelling using Object-Oriented Pr/T-Nets.

24/97 *Lutz Priese, Yurii Rogojine, Maurice Margenstern.* Finite H-Systems with 3 Test Tubes are not Predictable.

23/97 *Peter Baumgartner (Hrsg.).* Jahrestreffen der GI-Fachgruppe 1.2.1 'Deduktionssysteme' — Kurzfassungen der Vorträge.

22/97 *Jens M. Felderhoff, Thomas Marx.* Erkennung semantischer Integritätsbedingungen in Datenbankanwendungen.

21/97 *Angelika Franzke.* Specifying Object Oriented Systems using GDMO, ZEST and SDL'92.

20/97 *Angelika Franzke.* Recommendations for an Improvement of GDMO.

19/97 *Jürgen Dix, Luís Moniz Pereira, Teodor Przymusinski.* Logic Programming and Knowledge Representation (LPKR '97) (Proceedings of the ILPS '97 Postconference Workshop).

18/97 *Lutz Priese, Harro Wimmel.* A Uniform Approach to True-Concurrency and Interleaving Semantics for Petri Nets.

17/97 *Ulrich Furbach (Ed.).* IJCAI-97 Workshop on Model Based Automated Reasoning.

16/97 *Jürgen Dix, Frieder Stolzenburg.* A Framework to Incorporate Non-Monotonic Reasoning into Constraint Logic Programming.

15/97 *Carlo Simon, Hanno Ridder, Thomas Marx.* The Petri Net Tools Neptun and Poseidon.

14/97 *Juha-Pekka Tolvanen, Andreas Winter (Eds.).* CAiSE'97 — 4th Doctoral Consortium on Advanced Information Systems Engineering, Barcelona, June 16-17, 1997, Proceedings.

13/97 *Jürgen Ebert, Roger Süttenbach.* An OMT Metamodel.

12/97 *Stefan Brass, Jürgen Dix, Teodor Przymusinski.* Super Logic Programs.

11/97 *Jürgen Dix, Mauricio Osorio.* Towards Well-Behaved Semantics Suitable for Aggregation.

10/97 *Chandrabose Aravindan, Peter Baumgartner.* A Rational and Efficient Algorithm for View Deletion in Databases.

9/97 *Wolfgang Albrecht, Dieter Zöbel.* Integrating Fixed Priority and Static Scheduling to Maintain External Consistency.

8/97 *Jürgen Ebert, Alexander Fronk.* Operational Semantics of Visual Notations.

7/97 *Thomas Marx.* APRIL - Visualisierung der Anforderungen.

6/97 *Jürgen Ebert, Manfred Kamp, Andreas Winter.* A Generic System to Support Multi-Level Understanding of Heterogeneous Software.

5/97 *Roger Süttenbach, Jürgen Ebert.* A Booch Metamodel.

4/97 *Jürgen Dix, Luis Pereira, Teodor Przymusinski.* Prolegomena to Logic Programming for Non-Monotonic Reasoning.

3/97 *Angelika Franzke.* GRAL 2.0: A Reference Manual.

2/97 *Ulrich Furbach.* A View to Automated Reasoning in Artificial Intelligence.

1/97 *Chandrabose Aravindan, Jürgen Dix, Ilkka Niemelä .* DisLoP: A Research Project on Disjunctive Logic Programming.

1996

28/96 *Wolfgang Albrecht.* Echtzeitplanung für Alters- oder Reaktionszeitanforderungen.

27/96 *Kurt Lautenbach.* Action Logical Correctness Proving.

26/96 *Frieder Stolzenburg, Stephan Höhne, Ulrich Koch, Martin Volk.* Constraint Logic Programming for Computational Linguistics.

25/96 *Kurt Lautenbach, Hanno Ridder.* Die Lineare Algebra der Verklemmungsvermeidung — Ein Petri-Netz-Ansatz.

24/96 *Peter Baumgartner, Ulrich Furbach.* Refinements for Restart Model Elimination.

23/96 *Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, Wolfgang Nejdl.* Tableaux for Diagnosis Applications.

22/96 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE in Practice: a Case for KOGGE.

21/96 *Harro Wimmel, Lutz Priese.* Algebraic Characterization of Petri Net Pomset Semantics.

20/96 *Wenjin Lu.* Minimal Model Generation Based on E-Hyper Tableaux.

19/96 *Frieder Stolzenburg.* A Flexible System for Constraint Disjunctive Logic Programming.

18/96 *Ilkka Niemelä (Ed.).* Proceedings of the ECAI'96 Workshop on Integrating Nonmonotonicity into Automated Reasoning Systems.

17/96 *Jürgen Dix, Luis Moniz Pereira, Teodor Przymusinski.* Non-monotonic Extensions of Logic Programming: Theory, Implementation and Applications (Proceedings of the JICSLP '96 Postconference Workshop W1).

16/96 *Chandrabose Aravindan.* DisLoP: A Disjunctive Logic Programming System Based on PROTEIN Theorem Prover.

15/96 *Jürgen Dix, Gerhard Brewka.* Knowledge Representation with Logic Programs.

14/96 *Harro Wimmel, Lutz Priese.* An Application of Compositional Petri Net Semantics.

13/96 *Peter Baumgartner, Ulrich Furbach.* Calculi for Disjunctive Logic Programming.

12/96 *Klaus Zitzmann.* Physically Based Volume Rendering of Gaseous Objects.

11/96 *J. Ebert, A. Winter, P. Dahm, A. Franzke, R. Süttenbach.* Graph Based Modeling and Implementation with EER/GRAL.

10/96 *Angelika Franzke.* Querying Graph Structures with G^2QL.

9/96 *Chandrabose Aravindan.* An abductive framework for negation in disjunctive logic programming.

8/96 *Peter Baumgartner, Ulrich Furbach, Ilkka Niemelä .* Hyper Tableaux.

7/96 *Ilkka Niemelä, Patrik Simons.* Efficient Implementation of the Well-founded and Stable Model Semantics.

6/96 *Ilkka Niemelä .* Implementing Circumscription Using a Tableau Method.

5/96 *Ilkka Niemelä .* A Tableau Calculus for Minimal Model Reasoning.

4/96 *Stefan Brass, Jürgen Dix, Teodor. C. Przymusinski.* Characterizations and Implementation of Static Semantics of Disjunctive Programs.

3/96 *Jürgen Ebert, Manfred Kamp, Andreas Winter.* Generic Support for Understanding Heterogeneous Software.

2/96 *Stefan Brass, Jürgen Dix, Ilkka Niemelä, Teodor. C. Przymusinski.* A Comparison of STATIC Semantics with D-WFS.

1/96 *J. Ebert (Hrsg.).* Alternative Konzepte für Sprachen und Rechner, Bad Honnef 1995.

1995

21/95 *J. Dix and U. Furbach.* Logisches Programmieren mit Negation und Disjunktion.

20/95 *L. Priese, H. Wimmel.* On Some Compositional Petri Net Semantics.

19/95 *J. Ebert, G. Engels.* Specification of Object Life Cycle Definitions.

18/95 *J. Dix, D. Gottlob, V. Marek.* Reducing Disjunctive to Non-Disjunctive Semantics by Shift-Operations.

17/95 *P. Baumgartner, J. Dix, U. Furbach, D. Schäfer, F. Stolzenburg.* Deduktion und Logisches Programmieren.

16/95 *Doris Nolte, Lutz Priese.* Abstract Fairness and Semantics.

15/95 *Volker Rehrmann (Hrsg.).* 1. Workshop Farbbildverarbeitung.

14/95 *Frieder Stolzenburg, Bernd Thomas.* Analysing Rule Sets for the Calculation of Banking Fees by a Theorem Prover with Constraints.

13/95 *Frieder Stolzenburg.* Membership-Constraints and Complexity in Logic Programming with Sets.

12/95 *Stefan Brass, Jürgen Dix.* D-WFS: A Confluent Calculus and an Equivalent Characterization..

11/95 *Thomas Marx.* NetCASE — A Petri Net based Method for Database Application Design and Generation.

10/95 *Kurt Lautenbach, Hanno Ridder.* A Completion of the S-invariance Technique by means of Fixed Point Algorithms.

9/95 *Christian Fahrner, Thomas Marx, Stephan Philippi.* Integration of Integrity Constraints into Object-Oriented Database Schema according to ODMG-93.

8/95 *Christoph Steigner, Andreas Weihrauch.* Modelling Timeouts in Protocol Design..

7/95 *Jürgen Ebert, Gottfried Vossen.* I-Serializability: Generalized Correctness for Transaction-Based Environments.

6/95 *P. Baumgartner, S. Brüning.* A Disjunctive Positive Refinement of Model Elimination and its Application to Subsumption Deletion.

5/95 *P. Baumgartner, J. Schumann.* Implementing Restart Model Elimination and Theory Model Elimination on top of SETHEO.

4/95 *Lutz Priese, Jens Klieber, Raimund Lakmann, Volker Rehrmann, Rainer Schian.* Echtzeit-Verkehrszeichenerkennung mit dem Color Structure Code — Ein Projektbericht.

3/95 *Lutz Priese.* A Class of Fully Abstract Semantics for Petri-Nets.

2/95 *P. Baumgartner, R. Hähnle, J. Posegga (Hrsg.).* 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods — Poster Session and Short Papers.

1/95 *P. Baumgartner, U. Furbach, F. Stolzenburg.* Model Elimination, Logic Programming and Computing Answers.