Appligraph
Applications of Graph Transformation

ESPRIT Working Group No. 22565

Second Annual Progress Report

Hans-Jörg Kreowski and Detlef Plump
(editors)

April 1999

Contents

1 Introduction

This report summarizes the activities in the second year of the ESPRIT Working Group APPLIGRAPH, covering the period from April 1, 1998, to March 31, 1999. The principal objective of this Working Group is to promote applied graph transformation as a rule-based framework for the specification and development of systems, languages, and tools and to improve the awareness of its industrial relevance.

The work in APPLIGRAPH is organized according to the following three research ares:

- *Language Issues*
- *Tool Issues*
- *Application Domains*

In the area of *Language Issues*, features in graph transformation based languages are investigated, like typing, modularity, refinement, parallelism, concurrency, distribution, optimisation, and correctness.

The area of *Tool Issues* is concerned with the conception and design of support tools for graph transformation based languages including editors, parsers, interpreters, compilers, optimisers, verifiers, and graphical user interfaces.

In *Application Domains*, the usefulness of graph transformation is demonstrated by case studies in various areas like definition of visual languages, database models, concurrent and distributed systems, software process modelling, and implementation of programming languages.

2 Members and Scientific Correspondents

The following research teams participate in APPLIGRAPH:

University of Technology Aachen: Manfred Nagl (team leader), Andreas Winter, Bernhard Westfechtel, Carl-Arndt Krapp, Ansgar Radermacher, Katja Cremer, Stefan Gruner, Peter Heimann, Andre Deparade, Anita Wagner, Roland Baumann, Peter Klein, Florian Sauer, Dirk Jäger, Manfred Münch

University of Antwerp: Dirk Janssens, Jan Paredaens (team leaders), Jan Van den Bussche, Nico Verlinden

Technical University of Berlin: Hartmut Ehrig, Herbert Weber (team leaders), Martin Grosse-Rhode, Julia Padberg, Gabi Taentzer, Roswitha Bardohl, Manuel Koch, Boris Melamed, Michael Rudolf, Mercè Llabrés-Segura

University of Bremen: Otthein Herzog (team leader), Hans-Jörg Kreowski (team leader and Working Group coordinator), Detlef Plump (Working Group manager), Frank Drewes, Martin Gogolla, Berthold Hoffmann, Gregor Joeris, Renate Klempien-Hinrichs, Peter Knirsch, Sabine Kuske, Sima Meamar-Nobari, Ubbo Visser

University of Leiden: Grzegorz Rozenberg (team leader), Egbert Boers, Joost Engelfriet, Jetty Kleijn, Vincent Zweije, Giorgio Busatto

University of Nijmegen: Rinus Plasmeijer (team leader), Marko van Eekelen, Sjaak Smetsers, Erik Barendsen, John van Groningen, Peter Achten, Marco Pil, Pascal Serrarens, Ronny Wichers Schreur, Arjan van Yzendoorn, Pieter Koopman, Thorsten Zörner, Maarten de Mol

University of Paderborn: Gregor Engels (team leader), Reiko Heckel, Annika Wagner

University of Pisa: Ugo Montanari (team leader), Paolo Baldan, Stefano Bistarelli, Roberto Bruni, Andrea Corradini, Andrea Maggiolo-Schettini, Marco Pistore, Francesca Rossi, Jürgen Müller

University of Rome: Francesco Parisi-Presicce (team leader), Stefano Levialdi, Piero Mussio, Luigi Cinque, Paolo Bottoni, Marta Simeoni

Appligraph corresponds and cooperates with other experts in the field and related applied areas. The list of scientific correspondents includes: Virginio Cantoni (Pavia, Italy), Maria Francesca Costabile (Bari, Italy), Heiko Dörr (Daimler-Benz AG, Berlin, Germany), Jürgen Ebert (Koblenz, Germany), Michael Gödicke (Essen, Germany), Herbert Göttler (Mainz, Germany), Marc Gyssens (Limburg, Belgium), Annegret Habel (Oldenburg, Germany), T. Harju (Turku, Finland), Neil Jones (Copenhagen, Denmark), Simon Peyton Jones (Glasgow, Scotland), Michael Löwe (Hannover, Germany), Alfonso Pierantonio (L'Aquila, Italy), Wilhelm Schäfer (Paderborn, Germany), Hans-Jürgen Schneider (Erlangen, Germany), Andy Schürr (München, Germany), Gabriel Valiente Feruglio (Barcelona, Spain), Albert Zündorf (Paderborn, Germany)

3 Summary of Activities

3.1 Research

During the second year of APPLIGRAPH, research has been carried out according to the Working Group Programme as specified in Annex I of the contract. The results of this research are documented in 22 papers published or accepted for publication in journals and books, 55 papers published or accepted for publication in conference and workshop proceedings, 3 dissertations and 8 technical reports. See Section 4 for a description of these results, and the list of references at the end of this report.

3.2 Meetings

The International Workshop on Graph Transformation, a Business Meeting, a Team Leader Meeting and two Subgroup Meetings have been organized during the second year of the Working Group.

International Workshop TAGT '98. The Sixth International Workshop on Theory and Application of Graph Transformation (TAGT '98), organized by the Paderborn team, took place in Paderborn from November 16 to November 20, 1998. The workshop had 76 participants, among them 42 APPLIGRAPH members and 5 Scientific Correspondents of APPLIGRAPH. The scientific programme comprised 68 talks, partly in parallel sessions, and a panel discussion. One of the talks was an overview of the APPLIGRAPH Working Group, presented by the manager D. Plump.

Second Business Meeting. The 2nd APPLIGRAPH Business Meeting took place on November 17, 1998, during TAGT '98. This meeting had 18 participants. The coordinator H.-J. Kreowski summarized the objectives of the Working Group and the current state of affairs. Moreover, M. Nagl and A. Schürr reported on the preparation of the AGTIVE Symposium on Industrial Relevance of Graph Transformation (see below).

Third Team Leader Meeting. This meeting took place on November 19, 1998, during TAGT '98. Several administrative and financial issues, the information dissemination of the Working Group and the future major events and subgroup meetings were discussed.

Subgroup Meeting in Paderborn. Following TAGT '98, an APPLIGRAPH Subgroup Meeting on *Application Domains* took place on November 20, 1998, in Paderborn. There were 17 participants. The programme of the meeting consisted of some talks on various case studies in applied graph transformation, and a general discussion on applications.

Subgroup Meeting in Erlangen. A Subgroup Meeting on *Concepts for Graph Transformation Languages* took place on February 22/23, 1999, in Erlangen. There were 10 participants. The main topic was object-orientation and data abstraction for graph transformation languages, in particular for the language GRACE.

3.3 Future Events

The following future events of the Working Group are currently planned:

Subgroup Meeting in Aachen. On August 31, 1999, prior to the AGTIVE Symposium, there will be a Subgroup Meeting on *Design of Graph Transformation Tools* in Aachen.

International Workshop and Symposium AGTIVE. An International Workshop and Symposium *AGTIVE — Applications of Graph Transformation with Industrial Relevance* will take place in Kerkrade (The Netherlands) on September 1–3, 1999. This event is organized by M. Nagl and A. Schürr.

Subgroup Meeting in Bremen. In December 1999 there will be a Subgroup Meeting on *Verification* and *Visualization* in Bremen.

Workshop GRATRA 2000. This workshop will take place in Berlin, as a satellite event of the ETAPS 2000 conference (March 27 – April 1, 2000). The workshop will be a joint event with the TMR Research Network GETGRATS, and is organized by H. Ehrig and G. Taentzer.

3.4 Information Dissemination

In the second year of APPLIGRAPH, the Working Group continued its efforts to promote research activities in applied graph transformation and to improve the systematic exchange and dissemination of information on graph transformation and its applications. This includes the following activities:

Handbook of Graph Transformation. The work on the second and third volume of the *Handbook of Graph Grammars and Computing by Graph Transformation* has been finished. Both volumes are in press and will appear with World Scientific Publishers, see [EEKR99, RMEK99]. The second volume is on *Applications, Languages, and Tools* and the third is on *Concurrency, Parallelism, and Distribution.* The first volume on *Foundations* appeared two years ago [Roz97].

Selected Papers of TAGT '98. The Working Group supports the editing of a book with selected papers of the Sixth International Workshop on Theory and Application of Graph Transformation (TAGT '98), which will appear in the series *Lecture Notes in Computer Science* of Springer-Verlag.

APPLIGRAPH NEWS. An *Electronic Bulletin Board* has been created by which APPLIGRAPH members distribute paper announcements and related news via the Internet. Technically this is realized by using the existing GRAGRA mailing list, which is maintained by the Aachen group. The e-mail address for distributing news is `gragra@bach.informatik.rwth-aachen.de`.

WWW Homepage. The Working Group's WWW homepage (`http://www.informatik.uni-bremen.de/grp/ag-ti/appligraph/home.html`) contains information about the project (proposal, synopsis, progress reports) and links to related information (workshops, graph transformation tools, graph grammar bibliograhy and graph grammar mailing list).

4 Scientific Results of the Second Year

The following description of the second year's scientific achievements is structured according to three research areas of the Working Group Programme: *Language Issues*, *Tool Issues*, and *Application Domains.*

4.1 Language Issues

4.1.1 Modularity

In [AEH+99] some recent developments in applying graph transformation as a rule-based framework for the specification and development of systems, languages, and tools are presented. After reviewing the basic features of graph transformation, a selection of applications is discussed, including the evaluation of functional expressions, the specification of an interactive graphical tool, an example specification for abstract data types, and the definition of a visual database query language. The case studies indicate the need for suitable structuring principles which are independent of a particular graph transformation approach. To this end, the concept of transformation units is presented, which allows systematic and structured specification and programming based on graph transformation.

In [KK99] the notions of transformation unit and transformation module are introduced as means for constructing large graph transformation systems from small ones in a structured and systematic way. A transformation unit comprises a set of rules, descriptions of initial and terminal graphs, a control condition, and a set of imported transformation units. Semantically, it transforms initial graphs to terminal ones by interleaving ordinary direct derivation steps with calls of imported transformation units such that the control condition is obeyed. Putting transformation units together yields transformation modules. A module may have an import interface consisting of formal parameter units and an export interface consisiting of the units that are made available to the environment. The formal parameter units can be instantiated by exported units of other modules. The introduced framework is independent of a particular graph transformation approach and, therefore, it may enhance the usefulness of graph transformations in many contexts.

In [HHKK98] a module concept for graph transformation is presented which encapsulates rules and graph transformation units, and exports some of them. Each module may import a set of graph transformation units and rules from other modules. The local transformation units define compound graph transformations that call rules and other transformation units which are defined or imported by the surrounding module. Moreover, they allow pre- and post-conditions to be specified. Semantically, each module specifies a binary relation on graphs for each exported item. The transformation units of modules are a simplified version of those in [KK99], because they are relieved from their interfaces and local rules.

One of the basic components of a transformation unit is its control condition which allows to restrict the nondeterminism of graph transformation. The concept of transformation units is abstract in the sense that each formalism which specifies a binary relation on graphs can be used as a control condition. In [Kus98] a selection of concrete classes of control conditions is discussed which seem to provide reasonable expressive power for specifying and programming with transformation units. These include regular expressions, once, as-long-as-possible, priorities, and conditionals. It is shown which classes of control conditions can be replaced by others without changing the semantics of the corresponding transformation unit. Moreover, three properties of control conditions are studied: minimality, invertibility and continuity.

Graph transformation systems support the formal modeling of dynamic, concurrent, and distributed systems. States are given by their graphical structure, and transitions are modeled by graph transformation rules. Refinements are the basic steps in the development of graph transformation system specifications. Starting from an abstract description of the system's behaviour stepwise refinements yield more and more concrete specifications, that should finally be directly implementable on a machine. For graph transformation systems, a refinement of a more abstract specification by a more concrete one is given by associating with each rule of the more abstract specification a composition of rules of the more concrete one. The basic composition operations for graph transformation rules are sequential composition by concatenation and parallel composition by amalgamation, i.e refinement in time (temporal refinement) and space (spatial refinement).

In [GSP98], these two kinds of refinement relations for graph transformation systems are investigated in order to support the development of a module concept for graph transformation systems. In a spatial refinement a transformation rule is refined by an amalgamation of rules while in a temporal refinement it is refined by a sequence of rules: in both cases, the refinement relation supports the modeling of implementation. In the first part of [GSP99], the properties of spatial and temporal refinements are further investigated while, in the second part, they are employed for the development of a module concept for typed graph transformation systems. Finally, as a first step towards an algebra of modules, the operations of union and composition of modules are introduced.

In [GSP98] these two operations are separated while in [GPS98] they can be combined arbitrarily: a language of expressions has been introduced to syntactically describe their composition.

Refinements have been employed for introducing a module concept for graph transformation systems. A module consists of three specifications: two interfaces, import and export, and a body that implements the services offered by the export interface, possibly using the features required at the import interface. The export is related to the body by a refinement that defines the implementation, whereas the import interface is simply a subsystem of the body, meaning that the external features required by the import interface are simply included in the body. Modules have been defined in [GSP99] w.r.t. the separated spatial and temporal refinements, and then the notion has been rephrased in [GPS98] for refinements via rule expressions.

In [HEET98], a systematic approach is introduced for classifying and comparing modularity concepts that have been proposed for graph transformation systems. The approach is based on the following observation: Modules and module interconnections consist of basic specifications and relations between such specifications. Hence, a natural approach for characterizing a module concept is to answer the following three questions: (1) What are the basic specifications? (2) Which relations between specifications are used? and (3) How are specifications and relations combined to modules and interconnections? In [HEET98], these questions are answered for five different module concepts for graph transformation systems.

The development of modularity aspects of ESM and Actor grammars has continued along various lines. The scope of the research has broadened in the sense that a general framework for embedding-based graph rewriting, called Local Action Systems (LAS), has been proposed [Jan99] and that the modularity concept is now developed for this general framework. In the LAS approach, productions are viewed as composed of local actions. When applied to actor systems, this corresponds to the fact that an event (actor grammar production) is composed of more basic instructions, combined into a script. Thus productions as such embody already a rather primitive form of modularity. The LAS approach has the particular feature that each process, composed of productions, corresponds to a new production. Hence a step, or production occurrence, of a high-level system description can be refined into a process consisting of a combination of lower level steps. A module concept for LAS systems is being developed based on these ideas. In broad lines, the approach is comparable to that of Transformation Units, but it is based on a process semantics, rather than on relations.

In [Hec98] a new technique for the specification of concurrent and reactive systems is developed which combines the rule-based approach of graph transformation with the idea of under-specified computations in reactive subsystems. Classical rule-based systems come with an *implicit frame condition*, that is, nothing has to change beyond what is explicitly specified. Thereby, they assume to have complete control about the system and its evolution. Hence, the interaction between the system and its (possibly unknown) environment, which is essential for reactive systems, can not be explicitly modeled. *Open graph transformation systems* extend graph transformation systems (in the double-pushout approach) by a new loose semantics for rule-based systems, which allows to model interaction, and by *explicit frame conditions* which restrict the allowed interactions to an interface of *open types*. In this way, they support the graphical modeling and formal specification of views as reactive subsystems. For specifying dynamic properties of open graph transformation systems, the approach is integrated with a temporal logic based on regular expressions. Based on the loose semantics of views as reactive subsystems, such properties can be verified in a compositional way. The concepts are validated by two case studies; a Banking system exemplifying the graphical modeling of views and their integration, and a formal specification and view-based verification of a mutual exclusion algorithm.

In [CGRH99], the development of an algebra of graph derivations as a descriptive model of graph transformation systems is discussed. For that purpose, a categorical three level approach for the construction of models of computations based on structured transition systems is used. Categorically the algebra of graph derivations can then be characterized as a double category with finite horizontal colimits, which is freely generated by a suitable representation of the graph transformation system as a structured transition system. Therefore this semantics is compositional with respect to operations on graph transformation systems expressible in terms of colimits.

4.1.2 Parallelism and Concurrency

The use of ESM and related graph transformation systems has been investigated both on the level of the semantics of concurrent languages in general and on the level of the more specific case of actor systems. The ESM model has a number of ad-hoc features that have been removed by generalizing them to Local Action Systems. The latter allow a simple and flexible notion of a process [Jan99]. Similar to what is the case for Petri net processes, configurations that occur in the execution of a process can be easily characterized, and consequentely a process uniquely determines a transformation between graphs. In the particular case of Actor grammars, their formulation in terms of LAS allows one to model the individual instructions of an actor script as local actions. This is further evidence of the fact that this type of graph rewriting is well suited, not only for defining the semantics of actor systems, but also as a formalism for their (concurrent) implementation.

Building on the results published in [BCM98a, BCM98b] during the first year of the project, in [BCM99] the authors propose an unfolding semantics for graph grammars in the algebraic, double-pushout approach. The unfolding construction generates from a given typed grammar an *occurrence grammar*, i.e., a safe grammar satisfying suitable acyclicity constraints. Similarly to the case of Petri nets, each production of the unfolding represents an application of a production of the original grammar, and each item of its type graph represents an item generated during a derivation. A graphical, net-like representation of occurrence grammar is proposed, on which a “token game” can be played in an effective way. From the unfolding of a grammar an event structure can be extracted by considering a suitable partial order of configurations, which is a prime algebraic domain. This event structure semantics is shown to be equivalent to those proposed in [Sch94] and in [CEL$^+$96].

4.1.3 Optimization and Correctness

In [KV98] the notions of redundancy and subsumption—as known from the area of rule-based systems—are considered in the context of system verification. A rule is redundant if it can be removed without affecting the semantics of the system; it is subsumed by another rule if each application of the former one can be replaced by an application of the latter one with the same effect. Redundancy and subsumption are treated in the framework of high-level replacement systems, which generalize graph and hypergraph grammars. The main results of the paper are a characterization of subsumption and a sufficient condition for redundancy, which involves composite productions.

In [HMP98] three variants of the double-pushout approach to graph transformation are investigated. By considering injective matching and/or arbitrary right-hand morphisms in rules, three approaches are obtained that are more expressive than the traditional approach with arbitrary matching and injective right-hand morphisms. For each of the three variants, it is clarified whether the well-known commutativity and parallelism theorems are still valid. Modified results are given where necessary. In particular, for the most general approach with injective matching and arbitrary right-hand morphisms, sequential and parallel commutativity is established by strengthening sequential and parallel independence.

One of the basic concepts for the transformation of graphs is node replacement, in particular the NCE approach. As graphs do not offer a sufficiently rich model for certain applications, NCE rewriting has been generalized to directed hypergraphs where an edge may be incident to an arbitrarily long sequence of nodes. At the same time, pullback rewriting has been proposed as a generic mechanism for node replacement which allows to study this concept independently of any concrete graph (or hypergraph) model. [JK98] shows how node rewriting in directed hypergraphs fits in the abstract framework of the pullback approach.

The paper [GHK98] is concerned with the combination of graph transformations and temporal logic. The combination allows for the specification of dynamic properties of systems within the formalism of graph transformation. A graph-based semantics is given and their equivalence to the classical semantics is shown. The equivalence provides the use of already existing analysis techniques in the field of temporal logic. Furthermore, the construction of a minimal model is introduced allowing for the use of model checking.

The contents of the following papers have been described in the First Annual Progress Report: [Dre98a, DE98, Plu98].

4.1.4 Language Design

Why not combining two successful principles in one formalism: rules and graphs? This is the common vision of the graph transformation community and the main motivation behind the development of PROGRES, a visual language that supports PROgramming with Graph REwriting Systems. The handbook chapter [SWZ99] documents the development of the PROGRES language and its accompanying programming environment.

Recently developed PROGRES specifications have a typical size of more than one hundred pages. Keeping these specification in a consistent state and reusing parts of one specification within others is not possible without a module concept. This observation is also true for other graph transformation approaches, which suffer from the lack of any module concept. The paper

[SW98b] introduces a module concept for the graph rewriting (transformation) language PROGRES, which is closely related to the package concept of the standardized object-oriented modeling language UML. It supports a variety of software design patterns including the construction of "Abstract Graph Types" and "Updatable Graph Views".

The Unified Modeling Language (UML) mentioned above is the first object-oriented modeling language with a useful modularization and information hiding concept. It supports nesting, import, and refinement of so-called packages. Although it has become OMG standard the definition of the underlying semantics is imprecise, ambiguous, and contradictory. The paper [SW98a] translates UML's informal package definition into predicate logic formulas and solves some open problems concerning the visibility of exported and imported modeling elements. Furthermore, the paper sketches how these formulas are transformed into a specification based on PROgrammed Graph REwriting Systems. The formal definition of the UML package concept was a necessary prerequisite for the development a precisely defined and UML-compatible PROGRES language module concept.

In [MSW98] a closely related language design topic is presented, the integration of static integrity constraints into the PROGRES language. These constraints check a specification's semantics at run-time. The paper discusses a number of important design decisions such as when constraints have to be checked and what happens if constraint violations are detected. Active integrity constraints, which are well-known in the world of active database systems, enable the user to use an event-based programming style in PROGRES. In connection with modules, constraints are crucial means for programming by contract and for constructing updatable graph views.

The contents of the paper [ERT99] are described in the First Annual Progress Report.

4.2 Tool Issues

The Appligraph subarea *Tool Issues* comprises the activities of all partners concerning the design and realization of graph transformation tools. Such a topic is—of course—closely related to the topics of the other two subareas of Appligraph, *Language Issues* and *Application Domains*. The development of graph transformation tools for specification and programming is unfeasible without the development of appropriate specification and programming languages. And once realized, tools have to be applied in order to get feedback about their usefulness for certain application domains.

In the following, the tool building activities of all Appligraph members and their scientific correspondents are listed. Due to the fact that almost all presented projects deal with the development of editors, parsers, and interpreters based on graph transformation technology, this section presents each project in a separate subsection instead of following the structure of the Working Group Programme.

4.2.1 The AGG Tool Set

Started in the beginning of 1997, the development of the graph transformation system AGG(for Algebraic Graph Grammar system) is an on-going activity, putting into practice the algebraic single-pushout approach [Löw93] with several extensions. Although retaining the name AGGfrom [LB93], the system is redesigned from scratch, merely taking over some ideas and concepts from its predecessor. The implementation is done purely in Java, a modern object-oriented, platform independent programming language. The AGGsystem provides a platform for realization and illustration of theoretical concepts based on the algebraic approach to graph transformation, and it aims to evolve into an environment for visual specification and programming.

Taentzer, Rudolf, and Ermel [ERT99] give a survey on AGG's design and development concepts, functionality, and use from the user's perspective. By examining some examples, they show how the "AGG-approach" of integrating conventional textual programming in Java with graph transformation may be employed to solve common programming tasks.

The paper [Rud98] presents a way to represent and solve the problem of graph matching—also known as the subgraph homomorphism problem—as a constraint satisfaction problem (CSP), opening up direct access to the large variety of research findings on optimized solution algorithms for CSPs. By decoupling the solution algorithm from the concrete graph model, the approach allows for variations of the model without affecting the algorithm. Furthermore, complementing the standard CSP definition, a query concept is introduced to allow abstract representation of concrete implementation properties for optimization purposes.

4.2.2 The DiaGen Tool Set

Diagrams are frequently used to represent or to communicate complicated situations. Depending on the problem domain, diagrams have a specific graphical representation, syntax, and semantics. Diagrams with common syntax and semantics make up a diagram class. Graphical editors are required in order to create or edit diagrams on a computer. Diagram editors are graphical editors that are tailored to a specific diagram class.

DiaGen (*Dia*gram Editor *Gen*erator) is a framework under development that supports the user who has to write diagram editors. *DiaGen* comes with a generator which creates an editor core from a formal specification that mainly describes diagram syntax and editor behavior by graph transformation.

Last year's research mainly tried to make *DiaGen* applicable to a wider range of diagram classes. This was achieved by extending the diagrams' internal model. The original model of a diagram has been a standard hypergraph. This simple diagram model has done a good job for diagram classes like Nassi-Shneiderman diagrams, flowcharts, or Message Sequence Charts. It has turned out to be too restrictive for diagram classes which make use of more general spatial relationships such as "component C intersects with component D"in the case of Venn diagrams. The hypergraph model has been extended in order to allow to represent such diagrams, too. Now, special hyperedges that represent spatial relationships instead of diagram components are also allowed in [Min98c]. Diagram editors that allow free-hand editing (i.e., in contrast to syntax-directed editing, the user can freely move and modify diagram components) have to detect these spatial relationships while the user is editing the diagram. This is the task of a "graphical scanner" which has to be efficient since detection of spatial relationships has to be performed during user interaction. Therefore, a general pattern recognition approach is unsuited. The papers [Min98a, Min98b] propose an efficient scanning procedure for this purpose.

The diagram class specification which is used by *DiaGen* mainly consists of a hypergraph grammar which describes the diagrams' syntax in terms of their hypergraph models. In previous work, only context-free hypergraph grammars with optional embedding rules [BMST99] have been used, which may be parsed efficiently. The papers [Min98a, Min98b] propose a parsing procedure for a much more general hypergraph grammar class. More recent work has investigated how to translate a diagram into an abstract representation which can be used, e.g., for diagram semantics. As before, diagrams are represented by hypergraphs which are created from the diagram by a graphical scanner. The syntax of these hypergraphs is specified by a hypergraph grammar again. However, this grammar is extended in such a way that a derivation of the hypergraph also derives a hypergraph which can be used as an abstract representation of the diagram.

4.2.3 The Fujaba Tool Set

The Fujaba project aims to provide a CASE tool that combines UML and Graph Grammars to a well defined object-oriented specification language, so-called Story diagrams [JZ98a]. Story diagrams utilize UML activity diagrams as visual control flow specification language. Instead of just using pseudo code, in story diagrams the activities contain graph rewriting rules. To facilitate the reading of these graph rewriting rules for object-oriented designers, our graph rewriting rules adapt the notation of UML collaboration diagrams.

The Fujaba environment provides code-generation facilities that allows to generate Java code from class diagrams and story diagrams [FNTZ98]. This Java code does not rely on a havy-weight library or database but only on some standard container classes, i.e. the Java Generic Library from Objectspace. Thus, the generated code is easily integrated with other system parts, e.g. a graphical user interface.

Current work focusses on a methodological use of story diagrams for the development of object-oriented applications. This starts with the refinement of use-cases via so-called story boards. Story boards have a sementics close to story diagrams. However, while story diagrams are executable formal specifications that allow a direct translation to an object-oriented programming language, story boards describe only possible scenarios. Thus, story boards are used to outline first ideas about the internal behavior of the desired system. In the next step the story boards are analysed and serve as a basis for the formal specification of the behavior of the desired system using story diagrams. In the final phase the Fujaba environment generates code from the story diagrams that is comined with a graphical user interface and the other system parts.

4.2.4 The PROGRES Tool Set

The programming environment accompanying PROGRES offers assistance for creating, analyzing, compiling, and debugging graph transformations as well as for rapid prototyping activities. Being an integrated set of tools with support for intertwining these activities, it combines the flexibility of interpreted languages with the safeness of compiled and statically typed languages.

Nowadays, PROGRES is used in various projects for generating rapid prototypes of graphical engineeering tools. The paper [Jäg99] describes a PROGRES-based approach for generating graphical, structure-oriented software engineering tools from graph-based specifications. The approach is based on the formal meta modeling of visual languages using graph rewriting systems. Besides the syntactic and semantic rules of the language, these meta models include knowledge from regarded application domains. This enables the resulting tools to provide the user with high level operations for editing, analysing and executing models. Tools are constructed by generating source code from the meta model of the visual language, which is written in the very high level programming language PROGRES. The source code is integrated into a framework which is responsible for the invocation of commands and the visualization of graphs.

RESMOD is another example of a tool which was developed using PROGRES. Its underlying resource management model for development processes handles human and computer resources in a uniform way, supports aggregation of resources into resource configurations, separates between plan resources (for planning) and actual resources (for execution), and takes care of multi-project management [KW98].

The Varlet project developes a database reengineering environment based an programmed graph rewriting systems. The Varlet environment allows to load the schema information of an existing relational database system and to enrich this information with additional design knowledge. Utilizing all available design knowledge, an object-oriented class hierarchy reflecting the relation data structures is derived semi-automatically. This object-oriented class hierarchy may then be enriched by further exploiting the additional object-oriented design concepts like aggregation and inheritance. In [JZ98b] we show the application of the programmed graph rewriting system PROGRES for the development of a database reengineering environment, the Varlet environment. Especially, we show how the underlying graph grammar theory is used to guarantee the consistency of the original relational database schema with the resulting object-oriented class hierarchy.

Still another application domain is studied in [Rad98, Cre99], the development of distributed software system applications and the reengineering of old legacy system code into distributed applications. Although today eased by middleware there is still a need for more sophisticated tools, which provide software engineers with additional means for solving these problems. The paper [Rad98] presents a tool that prepares monolithic software applications for distribution by applying a sequence of transformations. The final transformation steps insert code interacting with the middleware. All transformations are planned on the architecture of a software system. The architecture is naturally represented as a graph, the transformations are specified as graph rewriting steps by means of the PROGRES language.

The dissertation [Cre99] deals with the problem to prepare existing legacy applications for the reuse in a distributed environment. For this purpose existing applications have to be reversely engineered to get a description on a higher level of abstraction than the source code. The acquired information is stored in form of graphs (call graphs, control flow graphs etc.). The necessary structural changes are performed on the reverse engineering graphs with the help of PROGRES graph rewriting rules. The transformations on the graphs are connected with source code transformations to couple changes on the abstract graph level with changes on the concrete source code level.

Finally, [SNG99] addresses the topic to integrate graph-based tools using so-called coupled graph grammars. The paper gives an informal introduction to the specification of so-called "integration tools" and concludes with an outlook on the graph-grammatical specification of software tools for chemical engineering purposes. A more detailed sketch of the specification method to be applied for that purpose has been described in [Gru98]. The project context of chemical engineering supported by graph-grammatically specified software tools is presented in [SNSG98] and in [CNG99].

4.2.5 Further Tool Development Activities

A system called Treebag, which implements the idea of tree-based object generation (and in particular picture generation, see 4.3.1), is presented in [Dre98c, Dre98e] (see also http://www.informatik.uni-bremen.de/~drewes/treebag).

Some tool-oriented research has also been conducted in Antwerp on the usage of Actor systems. It has been known for some time that an Actor system can, in principle, be described by an actor grammar. However, there is obviously a considerable distance between the syntactic form of concrete actor languages on the one hand and that of actor grammars on the other hand. An attempt to bridge this gap has led to the definition of a simple actor language that allows the programming of meaningful algorithms, such as sorting or parsing, but that can also be automatically translated into a set of (parametrized) actor grammar productions. A preliminary version of a translator has been written. This is further evidence that a massively parallel implementation of the actor language is possible, based on the concurrent application of actor grammar productions.

Finally, the IPSEN project has to be mentioned which built the platform for the development of the already mentioned PROGRES environment (and other software engineering tools). The final state of this project, which started more than one decade ago, is documented in [KNS99]. It addressed the following research topic based on graph transformation technology: Since the coining of the term software engineering in the late 1960's, the research discipline of that name deals with the improvement of the productivity of the software development and maintenance process. As a crucial aid in achieving this goal, the process can be supported by tools relieving the developer from mechanizable, uncreative bookkeeping tasks and preventing him from making errors. A collection of such tools is commonly called a software development environment (SDE). The construction of SDEs is, again, a software development process. In the course of the IPSEN project, a powerful machinery has been developed for the construction of SDEs themselves. It combines a variety of reuse approaches based on context-free grammars as well as graph transformation systems as specification formalisms, generator tools, and a framework implementation.

4.3 Application Domains

4.3.1 Visual Languages

Graph transformation seems to be especially appealing to define the syntax of visual languages since visual languages have multi-dimensional representations which have to be coded into one-dimensional strings otherwise. Furthermore, graph transformation as visual formalism is used as a basis for a certain kind of rule-based visual languages. In [BMST99], both aspects of graph transformation applied to visual languages are considered. Graphs may be used to define visual sentences consisting of a set of visual symbols such as graphic primitives, icons or pictures, and a set of relations in between. Graph grammars are thus a natural means for defining a visual language in terms of its graph language. Using graph grammars for visual language definition, types of visual languages are tightly coupled with types of graph languages. Expressive power of a visual language type has to be considered carefully against efficiency of tools, e.g., when parsing visual languages. Context-free as well as different kinds of context-sensitive languages are considered in [BMST99]. Based on these concepts, two VL environment generating tools and languages—DiaGen and GenGEd—are presented that comprise generators for visual editors and parsers based on syntax definition by graph transformation. Graph transformation languages form a kind of very high-level visual programming languages, where the underlying data models are graphs allowing to represent any kind of complex, structured information at a convenient level of abstraction. The programming paradigm to manipulate such graph data models is graph transformation performed by applying graph rules. General purpose languages PROGRES and AGG as well as special purpose languages DiaGen and GenGEd are discussed and compared with each other.

In [Bar98], GenGEd, a visual definition tool for visual languages, is described. Given an alphabet and grammar rules of a specific visual language, GenGEd generates a syntax-directed graphical editor for this language. GenGEd as well as each generated editor is based on algebraic graph grammars. A sentence is given by a graph structure consisting of a logical (abstract syntax) and a visual level (layout informations). Both levels are connected by layout operations. Visual language rules are defined by graph grammar rules. The underlying logical structure, however, is hidden from the user, but it is essential for a formal presentation and manipulation of graphical structures on both levels. The editor operations are performed by a graph transformation machine working on the logical level, whereas a graphical constraint solver manages the layout the user works with. [BE98] presents the conceptual model of GenGEd.

[BKR98] discusses the use of GenGEd to construct an editor for a kind of graph grammars that can be used to specify reactive systems.

In [DK99] an introduction to picture generation by collage grammars is given, including context-free, table-driven, and context-sensitive derivation modes.

In [Dre98d] the concept of tree-based picture generation is introduced. This yields a unified framework for the generation of pictures that can be extended to other sorts of objects as well. In the paper it is shown that collage grammars, iterated function systems, chain-code grammars, and 0L-systems with turtle interpretation are equivalent to natural types of tree-based picture generators.

Context-free collage and chain-code grammars are compared with respect to their generative power in [Dre98b]. It is shown that context-free collage grammars based on similarity transformations can be simulated by context-free chain-code grammars, but that, in general, the language classes generated by both types of grammars are incomparable.

4.3.2 Concurrent and Distributed Systems

In [TFKV99] the visual design of distributed systems by distributed graph transformation is presented. The kernel of this technique focusses on a structured graph with two abstraction levels, the network and the local level, and its transformation. On the network level, the (possibly) dynamic topological structure of a distributed system is specified. The local level contains the graphical description of local object and data structures where parts of them may be replicated in remote network nodes. Those structures may evolve independently or synchronized with remote structures using a suitable interaction mechanism. Network activities as well as local actions are described rule-based. The advantage of using rules in this context is to avoid purely sequential or series-parallel control flow, since rules naturally support the design of multiple threads of control on the level of events without prescribing the order of execution.

For the illustration of the concepts presented we use a reference application, modeling distributed version control for any kind of document. This subject becomes the more important the more remote software development becomes popular. In contrast to server-client solutions where all documents are held centrally, the documents are really distributed over the involved sites in our case.

To support the design of local systems by different developers, local views are introduced in [FKT98a]. A local view on a distributed system contains one local system with its import and export interfaces and remote interfaces to which it may have connections. Two local systems can communicate synchronously and asynchronously. This communication is simply modeled by sequential application of local view rules. A synchronization is based on amalgamation of rules. Local view grammars can be composed to distributed system grammars. Their operational semantics is given by distributed transformation sequences.

The paper [KFV98] shows the idea of combining transaction models for distributed systems with transaction-based applications by means of distributed graph transformations. As we are able to model both transaction models and applications within the common formalism of distributed graph transformations, we can identify an interface for the combination. Over this interface, transaction models can be changed for applications without costly changes of the application. A distributed configuration management occuring in distributed software development is used as a sample application.

In [FKT98b], a new visual design technique for distributed object systems based on graph transformation is presented. It includes the graphical description of remote object interaction, object migration and replication, communication and synchronization as well as dynamic reconfiguration of distributed systems. The notation is built up on UML providing these concepts as a possible extension of UML. Using graph transformation as underlying formal framework, basic consistencies in distributed systems can be ensured.

In [GMT98], the application of distributed graph transformation to the problem of specifying dynamic change in distributed systems is considered. Change in distributed systems is related to at least two levels. One is the management of change in a local node of the distributed system and how such a local change is then propagated to those nodes in the systems that need to know about the change. The other aspect is changing the structure of the distributed system itself. This implies, e.g., to add a local node or an entire subsystem to the distributed system or to remove it from the system. In some important application areas such operations must be done during runtime without disturbing the unmodified rest of the distributed computing system. An example—a ring database—shows how our approach can be applied to a small but nontrivial distributed system. This example shows nicely how the two aspects of change can be described uniformly using graph transformation rules and how the interaction of the two change aspects can be defined in an adequate way.

In [HK98], node replacement and hyperedge replacement are combined to a concept of atom replacement that allows the replacement of the atomic units in hypergraphs, that is, nodes and hyperedges can be replaced with hypergraphs. This concept may be seen as a framework for a systematic investigation of Petri-net refinements which even allows to combine place and transition refinement techniques.

Research about the concurrency aspects of the ESM approach to graph rewriting has been concentrated on two goals : on the one hand, the further development of the process semantics of these systems, and on the other hand the use of this semantics for the compositional verification of protocols. The process semantics of ESM systems (which generalizes the one for Actor grammars) has been refined and generalized to so-called Local Action Systems (LAS), see [Jan99]. The graph grammar productions in a LAS are not atomic, but composed of simpler local actions. Since LAS are parametrized on monoids of node and edge labels, not only ESM or Actor systems correspond to particular choices of these monoids, but also other choices are possible, geared to particular applications. A specification and compositional proof of a data transport protocol has been given in [JV97], in terms of a variant of ESM systems. The use a suitable LAS for this kind of task may lead to a further simplification. The use of LAS or ESM graph transformations as specification language has the advantage that the embedding mechanism of these transformations allows the specification of properties that are not in a natural way input-output properties.

4.3.3 Computer-supported Cooperative Work

Several contributions to the semantic foundation of Computer-supported Cooperative Work have appeared whose formal basis is Tile Logic [GM99]. Tiles are rewrite rules with side-effects, and they are well-suited for modeling coordination languages, since they can be composed both statically and dynamically via possibly complex synchronization and work-flow mechanisms. By analogy with rewriting logic, the tile model has purely logical presentation, where tiles are considered as suitable sequents subject to simple inference rules. The basic tile model presented in [GM99] is extended in [BMM98c, Bru99], where it is shown how to deal with configurations and observations relying on common auxiliary structures (e.g., for tupling and projecting components).

In [FM97] a tile-based semantics for located CCS is proposed where the operational and abstract concurrent semantics described in the literature now descend from general, uniform notions. In [FM99] the results of [FM97] are fully discussed to outline a general approach to the analysis of process calculi. Such an approach is extended in [MT98] to characterize similarities and differences between actors and asynchronous π-agents on the basis of a common framework where the two paradigms can be embedded.

Since several existing languages support rewriting logic, the overall aim of [MM98, BMM98c] is to define an implementation of tile logic in rewriting logic, thus making tile specifications executable. Due to the great variety of issues involved, many results are separately presented. In [MM98, BMM98c], to prove the correctness of the implementation, the relationship between the categorical models of tile logic (as originally presented in [BMM99b]) and of rewriting logic is formalized using partial membership equational logic techniques. In [BMM98b] some useful rewriting strategies (e.g., discarding computations not allowed in the tile model) are defined to make the implementation effective. In [BMM98a, BMM99a] the methodology is applied to several case studies (e.g., finite and full CCS).

Finally, in [BM99] an higher-order extension of tile logic is discussed, which can provide a powerful type framework with applications to the area of reactive calculi with mobility and higher-order features.

4.3.4 Design and Implementation of Programming Languages

The contents of the paper [HP99] have been described in the First Annual Progress Report.

The group in Nijmegen has investigated the suitability of the functional language CONCURRENT CLEAN [KFS+98, PvE98] for the timely development and maintanance of quality software. The CLEAN system has the advantage that it runs on small machines and that it generates programs that run relatively fast. A very special feature of the system is that, due to uniqueness typing, interfacing the pure functional language CLEAN with the imperative world can be done in a flexible and efficient way.

Although the existing CLEAN compiler is very reliable and very fast, it was very difficult to add extensions to the system. This is due to the fact that the compiler was incrementally written in C over many, many years. Therefore it was necessary to re-code a huge part of the CLEAN compiler, this time in CLEAN of course. From the old compiler only the back-end will be reused: the code generation part generating platform independent ABC-code (comparable to byte code) and the strictness analyzer (based on abstract reduction, see Nocker's PhD thesis). The main advantage of this implementation is that new extensions to CLEAN can be made much easier, also by people who are less experienced. The Nijmegen group has managed to re-implement most of the functionality of the old compiler. In addition a number of extensions have been added: (1) The type class system of CLEAN is extended with multi-parameter type constructor classes to enable the specification of programs with a large freedom of choice of the concrete data structures being used. (2) The basic primitives for dynamic typing has been added. (3) New scope rules for modules that ease the redefinition of pre-defined functions and types. (4) Cyclic module dependencies are permitted.

A tool for interfacing CLEAN with C has been designed and implemented. It has been used to interface the new CLEAN 2.0 compiler with the existing 1.3 backend written in C. A definition of the object I/O library has been frozen and it has been implemented on the Windows and Macintosh platforms. A programmers tutorial report has been written and made available on the internet. Library functions to support platform independent printing have been designed and implemented.

The dynamic type system is further refined and can deal with types as found in the static type system (synonym types, abstract types, existential types). The integration of static and dynamic typing has been improved. Using a special kind of constructor classes the type restrictions imposed by the static type of the environment can be translated into a type requirement of the dynamic type [Pil99]. The "Dynamics" are almost implemented in the new CLEAN 2.0 compiler.

An explicit message passing mechanism has been added to CONCURRENT CLEAN [Ser99, Ser98]. It provides features which were not (fully) supported. These are: data-driven communication, multicasting messages, non-determinism and inter-program communication. We provide an easy-to-use construction with channels and send/receive functions on them. In addition to this we introduced explicit threads which, in combination with explicit message passing, enabled us to write concise concurrent algorithms which can be executed with little overhead. We designed a library for general message passing, including synchronous, asynchronous, bi-directional, TCP/IP and broadcasting protocols. It is possible to write functions independent of the message passing protocol, enabling quick comparisons between protocols. Using the explicit threads and message passing we designed a concurrent version of the object i/o system, which enables us to write distributed, interactive applications. Because of the small overhead of message passing and the fact that the normal object i/o system simulates processes by doing its own scheduling, we improved the efficiency of interactive programs up to a factor 7.

4.3.5 Specification of Software Engineering Models

The idea of a combined reference model- and view-based specification approach has been proposed recently in the software engineering community. In [HEET99], a specification technique based on graph transformations is presented which supports such a development approach. The use of graphs and graph transformations supports an intuitive understanding and an integration of static and dynamic aspects on a well-defined semantical base. On this background, formal notions of view and view relation are developed and the behaviour of views is described by a loose semantics. A construction for automatic view integration is defined which assumes that the dependencies between different views are described by a reference model. The views and the reference model are kept consistent manually, which is the task of a model manager. All concepts and results are illustrated at the well-known example of a banking system.

Software development is a staged and evolutionary process. Multiple stakeholders with different needs and views collaborate to build a system from interoperating and heterogenous development artifacts. In such a setting, one has to cope with requirements changing dynamically during the entire lifetime of the system. Living with inconsistencies is natural within this changing world. Tool support is needed to tolerate inconsistencies and help developers to use them to drive the development process forward. In [GMT99], the application of distributed graph transformation to the problem of formalizing the integration of multiple perspectives in software development, called "ViewPoints", is considered. This work concentrates on requirements engineering. In [GMT99] it is demonstrated how inconsistencies management can be used as a tool for requirement analysis by presenting a sample integration of architecture design and performance evolution views.

Development of products in disciplines such as mechanical, electrical, or software engineering is a challenging task. Thus, sophisticated tools for supporting the management of development products, processes, and resources are urgently needed. The paper [Wes99] reports on the management system DYNAMITE which was designed and implemented in a joint project with mechanical engineers. The system was developed from a high-level formal specification based on programmed graph rewriting.

One specific aspect of the management system mentioned above is its ability to handle erroneous software development steps. While a software process is being executed, many errors and problems occur which require to reconsider previously executed process steps. In order to handle feedback in a process-centered software engineering environment, several requirements need to be addressed (adaptability, human intervention, impact analysis, change propagation, restoration of the work context, and traceability). Feedback management in DYNAMITE—discussed in [KSW98]—meets these requirements.

While the development of process modeling tools is one application area for graph transformations of particular interest, another one — the world wide web — might even represent the killer application for graph transformations, we are all looking for. The paper [BGM99] discusses the perspectives of applying the graph transformation approach of PROGRES [SWZ99] to to design structured natural-language documents. Examples of this kind are any documents written in markup languages such as HTML, SGML and XML. The chief benefits expected are a greater expressive power concerning the definition of the syntax of specific document classes and the rapid specification of application-specific functionality (in the form of document/graph transformations).

4.3.6 Term Graph Rewriting

Term graph rewriting is concerned with the representation of functional expressions as graphs, and the evaluation of these expressions by rule-based graph transformation. Representing expressions as graphs allows to share common subexpressions, improving the efficiency of term rewriting in space and time. Besides efficiency, term graph rewriting differs from term rewriting in properties like termination and confluence. A survey of term graph rewriting is given in [Plu99], where emphasis is given to the relations between term and term graph rewriting. The paper focusses on soundness of term graph rewriting with respect to term rewriting, on completeness for proving validity of equations and for computing term normal forms, on termination and confluence, and on term graph narrowing. The contents of the paper [AKP99] have been described in the First Annual Progress Report:.

In [CG98] the authors extend their previous work [CG97] on the categorical description of possibly cyclic term graph rewriting using suitable 2-categories to the rewriting of possibly cyclic term graphs. They show that this presentation is equivalent to the well-accepted operational definition proposed by Barendregt et. al., but for the case of *circular redexes*, for which they propose (and justify formally) a different treatment. The categorical framework is based on *traced gs-monoidal 2-categories*, and it allows to model also automatic garbage collection and rules for sharing/unsharing and folding/unfolding of structures. Furthermore, it can be used for defining various extensions of term graph rewriting, and for relating it to other rewriting formalisms.

4.3.7 Neural Networks

In [FKB98a], graph transformations are introduced as a unifying framework to formalize Neural Networks together with their training algorithms. The graph model for neurons and Neural Networks and graph transformation rules for basic algorithms are given. The benefits of this formal representation are discussed. Benefits treated are mainly the possibilities for proving properties of the training algorithms, as termination resp. convergence and equivalence of algorithms.

A Topology changing training algorithm for Probabilistic Neural Networks, more exactly the Dynamic Decay Adjustment (DDA) algorithm, is specified by graph transformation rules in [FKB98b]. With the help of concepts developed in the field of graph transformation, the termination and convergence of the DDA is shown.

In [FKB98c, FKB98d], graph transformations are used to show the equivalence of two training algorithms for Recurrent Neural Networks, namely Backpropagation Through Time and a variant of Real Time Backpropagation. Concepts of graph rewriting and term rewriting are used to show their equivalence. The proof also provides a whole class of related training algorithms. In part I of the papers the formalizations of the two training algorithms is shown, part II then shows the proof of the equivalence of both algorithms.

References

[AEH+99] Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34(1):1–54, 1999.

[AKP99] Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Bisimilarity in term graph rewriting. *Information and Computation*, 1999. To appear.

[Bar98] R. Bardohl. GenGEd - A Generic Graphical Editor for Visual Languages based on Algebraic Graph Grammars. In *Proc. IEEE Symposium on Visual Languages (VL'98)*, pages 48–55, 1998.

[BCM98a] P. Baldan, A. Corradini, and U. Montanari. Concatenable graph processes: relating processes and derivation traces. In *Proceedings ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[BCM98b] P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Proceedings FOSSACS'98*, volume 1378 of *LNCS*, pages 63–80. Springer Verlag, 1998.

[BCM99] P. Baldan, A. Corradini, and U. Montanari. Unfolding and event structure semantics for graph grammars. In W. Thomas, editor, *Proceedings FOSSACS'99*, volume 1578 of *LNCS*, pages 73–89. Springer Verlag, 1999.

[BE98] R. Bardohl and H. Ehrig. Conceptual model of the graphical editor GenGEd. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 32–39. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[BGM99] Anita Behle, Felix Gatzemeier, and Oliver Meyer. Graph technology for structured documents. In *Proceedings Conference on Software Engineering and Knowledge Engineering (SEKE 1999)*. IEEE Computer Society Press, 1999. To appear.

[BKR98] R. Bardohl and L. Korff-Ribeiro. Specifying a graph grammer editor for reachtive systems. In *Proc. Workshop Brasileiro de Metodos Formais*, Porto Alegre, Brazil, 1998.

[BM99] R. Bruni and U. Montanari. Cartesian closed double categories. In *Proceedings LICS'99*. IEEE Computer Society, 1999. To appear.

[BMM98a] R. Bruni, J. Meseguer, and U. Montanari. Implementing tile systems: Some examples from process calculi. In P. Degano, U. Vaccaro, and G. Pirillo, editors, *Proceedings 6th ICTCS'98*. World Scientific, 1998.

[BMM98b] R. Bruni, J. Meseguer, and U. Montanari. Internal strategies in a rewriting implementation of tile systems. In C. Kirchner and H. Kirchner, editors, *Proceedings 2nd WRLA'98*, volume 15 of *ENTCS*. Elsevier, 1998.

[BMM98c] R. Bruni, J. Meseguer, and U. Montanari. Process and term tile logic. Technical Report SRI-CSL-98-06, SRI International, 1998. Also Technical Report TR-98-09, Department of Computer Science, University of Pisa (1998).

[BMM99a] R. Bruni, J. Meseguer, and U. Montanari. Executable tile specifications for process calculi. In J.-P. Finance, editor, *Fundamental Approaches to Software Engineering (FASE'99)*, volume 1577 of *LNCS*, pages 60–76. Springer Verlag, 1999.

[BMM99b] R. Bruni, J. Meseguer, and U. Montanari. Symmetric and cartesian double categories as a semantic framework for tile logic. *MSCS*, 1999. To appear.

[BMST99] Roswitha Bardohl, Mark Minas, Andy Schürr, and Gabi Taentzer. Application of graph transformation to visual languages. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 3, pages 105–180. World Scientific, 1999. To appear.

[Bru99] R. Bruni. *Tile logic for synchronized rewriting of concurrent systems*. PhD thesis, Department of Computer Science, University of Pisa, 1999.

[CEL$^+$96] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An Event Structure Semantics for Graph Grammars with Parallel Productions. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Proceedings of the 5th International Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *LNCS*, pages 240–256. Springer Verlag, 1996.

[CG97] A. Corradini and F. Gadducci. A 2-categorical presentation of term graph rewriting. In *Proceedings CTCS'97*, volume 1290 of *LNCS*. Springer Verlag, 1997.

[CG98] A. Corradini and F. Gadducci. Rewriting on cyclic structures. Abstract presented at FICS'98, 1998.

[CGRH99] A. Corradini, M. Große-Rhode, and R. Heckel. An algebra of graph derivations using finite (co-)limit double theories. In *Recent Developments in Algebraic Development Techniques, Selected Papers*, volume 1589 of *LNCS*. Springer-Verlag, 1999.

[CNG99] K. Cremer, M. Nagl, and S. Gruner. Graph transformation based integration tools: Application to chemical process engineering. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 10, pages 369–394. World Scientific, 1999. To appear.

[Cre99] K. Cremer. *Graph-based Tools for Reverse Engineering and Reengineering (in German)*. PhD thesis, RWTH Aachen, 1999.

[DE98] Frank Drewes and Joost Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Information and Computation*, 145:1–50, 1998.

[DK99] Frank Drewes and Hans-Jörg Kreowski. Picture generation by collage grammars. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 11, pages 397–458. World Scientific, 1999. To appear.

[Dre98a] Frank Drewes. A characterization of the sets of hypertrees generated by hyperedge-replacement graph grammars. *Theory of Computing Systems*, 32:159–208, 1998.

[Dre98b] Frank Drewes. Some remarks on the generative power of collage grammars and chain-code grammars. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 425–432. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[Dre98c] Frank Drewes. Treebag—a tree-based generator for objects of various types. Report 1/98, Univ. Bremen, 1998.

[Dre98d] Frank Drewes. Tree-based picture generation. Report 7/98, Univ. Bremen, 1998.

[Dre98e] Frank Drewes. Treebag – Baum-basierte Generierung und Transformation von Objekten. In J. Dassow and R. Kruse, editors, *Proc. Informatik '98*, Informatik Aktuell, pages 47–56, 1998. In German.

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and*

Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools. World Scientific, 1999. To appear.

[ERT99] C. Ermel, M. Rudolf, and G. Taentzer. AGG-approach: Language and environment. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 14, pages 551–604. World Scientific, 1999. To appear.

[FKB98a] I. Fischer, M. Koch, and M. Berthold. Formalizing neural networks. In *Proceedings of the 5. GI-Workshop Fuzzy-Neuro-Systems '98*, Munich, 1998.

[FKB98b] I. Fischer, M. Koch, and M. Berthold. Proving properties of neural networks with graph transformations. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, 1998.

[FKB98c] I. Fischer, M. Koch, and M. Berthold. Showing the equivalence of two training algorithms — Part I. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, 1998.

[FKB98d] I. Fischer, M. Koch, and M. Berthold. Showing the equivalence of two training algorithms — Part II. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, 1998.

[FKT98a] I. Fischer, M. Koch, and G. Taentzer. Visual design of distributed object systems by graph transformation. Technical Report 98-15, Technical University of Berlin, 1998.

[FKT98b] Ingrid Fischer, Manuel Koch, and Gabi Taentzer. Local views on distributed systems and their communications. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 40–47. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[FM97] G. Ferrari and U. Montanari. Tiles for concurrent and located calculi. In C. Palamidessi and J. Parrow, editors, *Expressiveness in Concurrency*, volume 7 of *ENTCS*. Elsevier, 1997.

[FM99] G. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Information and Computation*, 1999. To appear.

[FNTZ98] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph grammar language based on the Unified Modelling Language and Java. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 112–121. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[GHK98] F. Gadducci, R. Heckel, and M. Koch. Model Checking Graph-Interpreted Temporal Formulas. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 292–299. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[GM99] F. Gadducci and U. Montanari. The tile model. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1999. To appear.

[GMT98] M. Goedicke, T. Meyer, and G. Taentzer. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 48–56. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[GMT99] M. Goedicke, T. Meyer, and G. Taentzer. Viewpoint-oriented software development by distributed graph transformation: Towards a basis for living with inconsistencies. In *Proc. Requirements Engineering 99*, 1999. To appear.

[GPS98] M. Große–Rhode, F. Parisi Presicce, and M. Simeoni. Refinements of graph transformation systems via rule expressions. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages

96–103. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[Gru98] S. Gruner. Why should chemo-engineers be interested in graph grammars? In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 72–79. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[GSP98] M. Große–Rhode, M. Simeoni, and F. Parisi Presicce. Spatial and temporal refinement of typed graph transformation systems. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proc. Mathematical Foundations of Computer Science (MFCS'98)*, pages 553 – 561. Springer LNCS 1450, 1998.

[GSP99] M. Große–Rhode, M. Simeoni, and F. Parisi Presicce. Refinements and modules for typed graph transformation systems. In *Recent Developments in Algebraic Development Techniques, Selected Papers*, volume 1589 of *LNCS*. Springer-Verlag, 1999.

[Hec98] R. Heckel. *Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems*. PhD thesis, TU Berlin, 1998.

[HEET98] Reiko Heckel, Hartmut Ehrig, Gregor Engels, and Gabriele Taentzer. Classification and comparison of modularity concepts for graph transformation systems. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 122–131. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[HEET99] Reiko Heckel, Hartmut Ehrig, Gregor Engels, and Gabriele Taentzer. A view-based approach to system modeling based on open graph transformation systems. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, pages 639–668. World Scientific, 1999. To appear.

[HHKK98] Reiko Heckel, Berthold Hoffmann, Peter Knirsch, and Sabine Kuske. Simple modules for GRACE. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 158–167. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[HK98] Annegret Habel and Renate Klempien-Hinrichs. Atom replacement in hypergraphs. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 182–189. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[HMP98] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 233–241. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[HP99] Annegret Habel and Detlef Plump. Complete strategies for term graph narrowing. In *Recent Developments in Algebraic Development Techniques, Selected Papers*, volume 1589 of *LNCS*. Springer-Verlag, 1999.

[Jäg99] Dirk Jäger. Generating tools from graph-based specifications. In *Proc. First Int. Symp. on Constructing Software Engineering Tools (CoSET'99)*. University of South Australia, School of Computer Science, 1999. To appear.

[Jan99] Dirk Janssens. Actor grammars and local actions. In Grzegorz Rozenberg, Ugo Montanari, Hartmut Ehrig, and Hans-Jörg Kreowski, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution*, chapter 2, pages 57–106. World Scientific, 1999. To appear.

[JK98] Hélène Jacquet and Renate Klempien-Hinrichs. Node replacement in hypergraphs: Translating NCE rewriting into the pullback approach. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 174–181. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[JV97] D. Janssens and N. Verlinden. Reasoning on graph-rewriting processes. Technical Report 97–07, University of Antwerp — UIA, Dept. of Math. and Comp. Sci., 1997.

[JZ98a] J.-H. Jahnke and A. Zündorf. Specification and implementation of a distributed planning and information system for courses based on story driven modelling. In *Proceedings of the Ninth International Workshop on Software Specification and Design*, pages 77–86. IEEE Computer Society, 1998.

[JZ98b] J.-H. Jahnke and A. Zündorf. Using graph grammars for building the varlet database reverse engineering environment. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 407–416. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[KFS+98] P.W.M. Koopman, J. Fokker, J.E.W. Smetsers, M.C.J.D. van Eekelen, and M.J. Plasmeijer. Functional programming in CLEAN. Report, University of Nijmegen, 1998. Available from www.cs.kun.nl/∼CLEAN.

[KFV98] M. Koch, I. Fischer, and V. Volle. Graphtransformation zur visuellen Beschreibung einer transaktionsgesteuerten Konfigurationskontrolle. *Rundbrief Datenbanken*, May 1998.

[KK99] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units and modules. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 15, pages 607–638. World Scientific, 1999. To appear.

[KNS99] P. Klein, M. Nagl, and A. Schürr. IPSEN tools. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by*

Graph Transformation, Volume 2: Applications, Languages, and Tools, chapter 5, pages 215–265. World Scientific, 1999. To appear.

[KSW98] Carl-Arndt Krapp, Ansgar Schleicher, and Bernhard Westfechtel. Feedback handling in software processes. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 417–424. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[Kus98] Sabine Kuske. More about control conditions for transformation units. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 150–157. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[KV98] Hans-Jörg Kreowski and Gabriel Valiente. Redundancy and subsumption in high-level replacement systems. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 233–241. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[KW98] Sven Krüppel and Bernhard Westfechtel. RESMOD: A resource management model for development processes. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 390–397. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[LB93] M. Löwe and M. Beyer. AGG — an implementation of algebraic graph rewriting. In *Proc. Fifth Int. Conf. Rewriting Techniques and Applications*, volume 690 of *LNCS*, pages 451–456. Springer-Verlag, 1993.

[Löw93] M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109:181–224, 1993.

[Min98a] Mark Minas. Automatically generating environments for dynamic diagram languages. In *Proc. 1998 IEEE Symposium on Visual Languages (VL'98)*, pages 70–71. IEEE Computer Society Press, 1998.

[Min98b] Mark Minas. Hypergraphs as a uniform diagram representation model. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 24–31. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[Min98c] Mark Minas. Specifying diagram languages by means of hypergraph grammars. In *Proc. Thinking with Diagrams '98 (TwD'98)*, pages 151–157, 1998.

[MM98] J. Meseguer and U. Montanari. Mapping tile logic into rewriting logic. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *LNCS*, pages 219–233. Springer Verlag, 1998.

[MSW98] M. Münch, A. Schürr, and A.J. Winter. Integrity Constraints in the Multi-Paradigm Language PROGRES. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 166–173. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[MT98] U. Montanari and C. Talcott. Can actors and π-agents live together? In A.D. Gordon, A.M. Pitts, and C. Talcott, editors, *Proceedings 2nd Workshop on Higher Order Operational Techniques in Semantics*, volume 10 of *ENTCS*. Elsevier, 1998.

[Pil99] M.R.C. Pil. Dynamic types and type dependent functions. In K. Hammond, T. Davie, and Ch. Clack, editors, *Proc. 10th Int. Workshop on Implementation of Functional Languages*, LNCS. Springer-Verlag, 1999. To appear.

[Plu98] Detlef Plump. Termination of graph rewriting is undecidable. *Fundamenta Informaticae*, 33(2):201–209, 1998.

[Plu99] Detlef Plump. Term graph rewriting. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 1, pages 3–62. World Scientific, 1999. To appear. Also Technical Report CSI-R9822, Computing Science Institute, University of Nijmegen, 1998.

[PvE98] M.J. Plasmeijer and M.C.J.D. van Eekelen. Language report Concurrent CLEAN — version 1.3. Report CSI-R9816, University of Nijmegen, 1998. Available from www.cs.kun.nl/∼CLEAN.

[Rad98] Ansgar Radermacher. Distribution of Applications with Graph Transformation Tools. Technical Report tr-ri-98-201, University of Paderborn, 1998.

[RMEK99] Grzegorz Rozenberg, Ugo Montanari, Hartmut Ehrig, and Hans-Jörg Kreowski, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution.* World Scientific, 1999. To appear.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations.* World Scientific, 1997.

[Rud98] Michael Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr-ri-98-201 in Reihe Informatik, pages 208–216. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[Sch94] G. Schied. On relating Rewriting Systems and Graph Grammars to Event Structures. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*, pages 326–340. Springer Verlag, 1994.

[Ser98] P.R. Serrarens. Using multicasting for optimising data-parallelism. In Ch. Clack, T. Davie, and K. Hammond, editors, *9th Int. Workshop on Implementation of Functional Languages*, volume 1467 of *LNCS*, pages 270–284. Springer-Verlag, 1998.

[Ser99] P.R. Serrarens. Explicit message passing for concurrent clean. In K. Hammond, T. Davie, and Ch. Clack, editors, *Proc. 10th Int. Workshop on Implementation of Functional Languages*, LNCS, pages 289–307. Springer-Verlag, 1999.

[SNG99] A. Schürr, M. Nagl, and S. Gruner. Integration tools supporting development processes. In M. Broy and B. Rumpe, editors, *Proc. RTSE'97 Workshop on Requirements Targeting Software and Systems Engineering*, LNCS 1526. Springer-Verlag, 1999. To appear.

[SNSG98] A. Schürr, M. Nagl, F. Sauer, and S. Gruner. Inkrementelle Integrationswerkzeuge für arbeitsteilige Entwicklungsprozesse. In M. Nagl and B. Westfechtel, editors, *Integration von Entwicklungssystemen in Ingenieur-Anwendungen*. Springer-Verlag, 1998.

[SW98a] A. Schürr and A.J. Winter. Formal Definition of UML's Package Concept. In M. Schader and A. Korthaus, editors, *The Unified Modeling Language — Technical Aspects and Applications*, pages 144–159. Physica-Verlag, Heidelberg, 1998.

[SW98b] A. Schürr and A.J. Winter. UML Packages for PROgrammed Graph REwriting Systems. In G. Engels and G. Rozenberg, editors, *Pre-Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number tr–ri–98–201 in Reihe Informatik, pages 132–139. Universität–Gesamthochschule Paderborn, Fachbereich Mathematik–Informatik, 1998.

[SWZ99] A. Schürr, A.J. Winter, and A. Zündorf. The PROGRES Approach: Language and Environment. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 13, pages 487–550. World Scientific, 1999. To appear.

[TFKV99] G. Taentzer, I. Fischer, M Koch, and V. Volle. Visual design of distributed systems by graph transformation. In Grzegorz Rozenberg, Ugo Montanari, Hartmut Ehrig, and Hans-Jörg Kreowski, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution*, pages 269–340. World Scientific, 1999. To appear.

[Wes99] Bernhard Westfechtel. Graph-based product and process management in mechanical engineering. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, chapter 9, pages 321–368. World Scientific, 1999. To appear.