Appligraph

Applications of Graph Transformation

ESPRIT Working Group No. 22565

First Annual Progress Report

Hans-Jörg Kreowski and Detlef Plump
(editors)

April 1998

Contents

1 Introduction

This report summarizes the activities in the first year of the ESPRIT Working Group Appligraph which started its work on April 1, 1997. 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), Andy Schürr, 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 Jaeger

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

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

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, Christoph Klauck, Renate Klempien-Hinrichs, Peter Knirsch, Sabine Kuske, Sima Meamar-Nobari

University of Leiden: Gregor Engels[1], Grzegorz Rozenberg (team leaders), Egbert Boers, Joost Engelfriet, Jetty Kleijn, Pieter Koopman, Vincent Zweije, Giorgio Busatto, Annika Wagner

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

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

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

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 (Hildesheim, 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), Gabriel Valiente Feruglio (Barcelona, Spain), Albert Zündorf (Paderborn, Germany)

3 Summary of Activities

3.1 Research

During the first year of Appligraph, research has been carried out according to the Working Group Programme as specified in Annex I of the contract.

[1]G. Engels moved to Univ. of Paderborn (Germany) by April 15, 1997

The results of this research are documented in 47 papers that have been published or are accepted for publication in journals and in conference and workshop proceedings, 2 dissertations and 15 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 first Appligraph Workshop, a Subgroup Meeting and two Team Leader Meetings have been organized during the first year of the Working Group. Moreover, the European School on Graph Transformation has been coorganized by the Working Group.

First Appligraph Workshop. This workshop, organized by the Bremen team, took place in Etelsen (near Bremen) on June 26/27, 1997. 25 Working Group members and 4 scientific correspondents participated in the meeting. Representatives of all Appligraph sites gave talks outlining the present and future research directions of their teams within the Working Group.

First Team Leader Meeting. This meeting took place on June 26, 1997, during the first Appligraph Workshop. Several administrative issues related to the start of the Working Group were discussed, and also the future major events of the project.

Subgroup Meeting. A first Subgroup Meeting on *Concepts for Graph Transformation Languages* took place on January 29/30, 1998, in Bremen. There were about 15 participants. The main topics of this meeting were module concepts and object-orientation for graph transformation languages like PROGRES and GRACE.

European School on Graph Transformation. This school took place March 2-7, 1998, in Bremen. It was organized by the TMR Research Network GETGRATS (General Theory of Graph Transformation Systems) and coorganized by the Appligraph Working Group. There were about 60 participants and 17 lecturers. The main purpose of this event was to introduce non-specialists to the field of graph transformation.

Second Team Leader Meeting. This meeting took place on March 3, 1998, during the European School on Graph Transformation. The main topics for discussion were budget issues and the future workshops and meetings of the Working Group.

3.3 Future Events

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

6th International Workshop. In November 1998, the *Sixth International Workshop on Theory and Application of Graph Transformations (TAGT'98)* will take place in Paderborn (Germany), organized by G. Engels. A Call for Papers (open not only for Working Group members) has already been distributed, see URL

http://www.uni-paderborn.de/cs/ag-engels/tagt98.html.

Subgroup Meeting on application domains. As a satellite event of the 6th International Workshop, there will be an APPLIGRAPH Subgroup Meeting in the research area Application Domains. The meeting will be organized by F. Parisi-Presicce.

International Workshop and Symposium on Industrial Relevance. An International Workshop and Symposium *AGTIVE — Applications of Graph Transformation with Industrial Relevance* shall take place in Kerkrade (The Netherlands) in September 1999. This event will be organized by the Aachen group.

Final workshop. The final workshop of the Working Group will take place in Berlin in 2000, as a joint event with the GETGRATS Research Network.

3.4 Information Dissemination

One of the aims of the Working Group is 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 is supported by the following activities:

Handbook of Graph Transformation. The first volume of the *Handbook of Graph Grammars and Computing by Graph Transformation*, edited by G. Rozenberg, has appeared with World Scientific Publishers (see [Roz97]). The Working Group currently works on two further volumes: one on *Applications, Languages, and Tools* and the other one on *Concurrency, Parallelism, and Distribution* (see [EEKR98] and [RMEK98]), both together covering the research areas of APPLIGRAPH.

Workshop proceedings. The Working Group will support the editing of the proceedings of the *Sixth International Workshop on Theory and Application of Graph Transformations (TAGT'98)* which are regularly published in the Springer LNCS series.

WWW site. The Working Group has installed a WWW site to inform about its activities. The URL is

http://www.informatik.uni-bremen.de/grp/ag-ti/appligraph/home.html.

In addition, this page contains a link to a page informing on existing graph transformation tools:

http://www-i3.informatik.rwth-aachen.de/private/andy/tools.html.

Description of the Working Group. A short description of the structure and the objectives of the Working Group has been published in the October '97 issue of the *Bulletin of the European Association for Theoretical Computer Science* (pages 109–114). The contribution also describes the TMR Research Network GETGRATS, and was jointly prepared by the Working Group Coordinator and A. Corradini, Coordinator of GETGRATS.

4 Scientific Results of the First Year

The following description of the first 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

Specification and rapid prototyping of graph manipulation software by using a graph transformation language such as PROGRES is a paradigm, which attracts more and more interests in various fields of computer science. Nowadays produced graph transformation specifications for process modelling tools or database query languages have a typical size of about 100 to 200 printed pages. They suffer severely from the lack of any graph transformation module concept. As a first remedy, a module concept in the flavor of Modula-2 has been developed for PROGRES. This module concept together with certain ideas how to use active database system technology for defining abstract graph views is published in [WS97].

Currently, the Modula-2 like module concept of PROGRES is refined and adapted to the Java-like package concept of the object-oriented modeling language standard UML. For this purpose it was necessary to study, formalize, and improve the package concept of UML. The results of these efforts are published in a short form for UML version 1.0 in [SW97] and in a more detailed form for UML version 1.1 in [SW98].

In [KKS97] a new concept for the modular development of rule-based graph transformation systems, called nested graph transformation unit, is presented. It consists of a (possibly cyclic) network of local graph transformation units that reflects the import structure of the system. It is shown that nested graph transformation units have a reasonable operational semantics, which is obtained by iterating the so-called interleaving operator.

In order to use a language based on graph transformations to develop large specifications and programs, it is necessary to develop notions of abstraction and modularity with structuring principals appropriate for graph transformation systems. The formal notion of implementation (and the corresponding abstraction) has been developed in [GRPPS97a, GRPPS97b] where each rule of a transformation system TS_1 is *refined* into an appropriate combination of *more elementary* rules of another transformation system TS_2. The refinements (temporal and spatial) are such that the applications of rules from TS_1 can be simulated by the applications of the corresponding refining rules in TS_2 (Preservation of Behavior). Control mechanisms are introduced in TS_2 in determining exactly how the refining rules must be used in order to realize the corresponding *abstract* rule in TS_1. These refinements are used in a notion of module [GRPPS98] which consists of three transformation systems, one representing the rules which are visible outside the module (EXP), one representing the implementation of such rules (BOD) and one representing the rules used in the implementation which can be subsequently refined by another module (IMP). The way the rules are implemented in BOD is hidden from the outside. Interconnection mechanisms, such as union and composition, have been defined for such modules in order to structure the overall TS in layers of abstraction.

The theory of algebraic graph transformation systems has been extended in [HEWC97c] by a *loose semantics*. This can be considered as a basis for the definition of a compositional semantics for systems made of many interconnected modules. For this purpose, graph transitions (defined using a double pullback) are introduced as a loose interpretation of graph productions. Moreover, each system is associated with a category of models (deterministic transition systems) defined as *coalgebras* over a suitable functor. Such category has a final object, which includes all finite and infinite transition sequences.

Building on these results, the aim of [HEWC97c] and [HEWC97b] is an extension of the theory of graph transformation systems in order to make them suitable for the specification of reactive systems. This is obtained by introducing *constraints* for graph transformation. The coalgebraic framework makes it possible to define a general notion of logic of behavioural constraints. Instances include, for example, start graphs, application and consistency conditions, and temporal logic constraints.

The idea of a combined reference model- and view-based specification approach has been proposed recently in the software engineering community. In [EHTE97] and [EEHT97], 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 being 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.

For the case of more than two views, more general scenarios are developed and discussed in [EEHT97].

An experimental actor language has been defined which is entirely implemented as a rule based system. In other words, the actor paradigm is used as a structuring technique for a graph rewriting system. In this context the main structuring concept, and thus the main tool for obtaining modularity, is the notion of an actor. Each actor class corresponds to a set of GTS production schemes, each of these schemes corresponds to a message type. The constructions developed for deriving abstract, compositional semantics from the basic process semantics of a GTS systems are applied to this particular case; particularly promising is the fact that it yields a way to express the fact that incoming messages of an actor are not arbitrary, because they can be sent only by actors that contain a reference to the accepting actor. This restriction has been notably hard to express by other methods.

4.1.2 Parallelism and Concurrency

In [BCM98a] an original notion of *concatenable graph processes* is proposed for typed graph grammars using the double-pushout approach. These structures allow the authors to relate in a completely satisfactory way the semantics based on derivation traces proposed by Corradini, Ehrig, Löwe, Montanari and Rossi in '94 and the notion of graph processes introduced by Corradini, Montanari, and Rossi in '96, showing their equivalence as truly concurrent semantics for graph grammars.

In [BCM98b] the authors propose an event based semantics for Contextual Place/Transition Petri, i.e., nets where resources that can be read without being consumed can be modeled as well. The result generalizes Winskel's seminal work on safe nets, making use of the original notion of Asymmetric Event Structure. These notions and results about contextual nets can be seen as a first step towards the definition of an unfolding semantics for graph grammars.

The paper [GM98a] lays the basis for a characterization of processes of *contextual* P/T nets as arrows of a suitable category. First, the class of *contextual processes* of a net is defined; then, it is shown how these processes can be modeled as arrows of a monoidal category, enriched with operators allowing for a controlled way to duplicate data.

The new approach of distributed graphs and graph transformation as presented in [Tae97] allows to use structured graph transformation on two abstraction levels, the network and the local level. The network level contains the description of the topological structures of a system. The local level covers the description of states and their transitions in local systems. Local state transitions may depend on others using suitable synchronization mechanisms. The main distribution concepts of categorical graph grammars presented by Schneider are combined with the algebraic approach to distributed graph transformation introduced by Ehrig et.al.

In [Koc97a], distributed graph transformation is extended to allow application conditions for distributed rules.

The development of a purely rule based (GTS based) semantics for a programming language yields interesting prospects for the exploitation of implicit concurrency because it offers a natural way to divide the work to be done into parallel tasks: each individual production occurrence acts on a small part of the global configuration. It has been shown that the GTS systems considered are confluent, so that the application of several productions to disjoint parts of a global configuration can proceed concurrently, without the need for explicit synchronization.

4.1.3 Optimization and Correctness

The problem whether a graph transformation system (in the "double pushout approach") terminates for all input graphs is undecidable. This is shown in [Plu98] by a reduction of the Post Correspondence Problem. As a consequence, total correctness of programs in graph transformation languages is in general undecidable, too.

In [DE98] the finiteness of ranges of a large class of tree transductions is proved to be decidable. It is shown that this yields a generalization of known decidability results concerning boundedness problems for functions on graphs and other domains. The functions that can be dealt with are definable by recursion on the derivation tree of a graph. This does not require a particular notion of graph rewriting.

In [Dre98, Dre97] the structure of hyperedge-replacement graph languages consisting of (hyper)trees is investigated. It is shown that these languages can be generated by productions whose right-hand sides are themselves hypertrees. However, for this one has to turn to sets of derivation trees produced by a slightly more powerful generation mechanism than usual, namely finite-copying top-down tree transducers. Thus, finite copying compensates exactly the loss in power that results from allowing only hypertrees as right-hand sides.

Constraints for graph transformation are introduced in [HEWC97c] and [HEWC97b] in order to restrict the - also in [HEWC97a] presented - loose semantics of graph transformation systems. A coalgebraic framework makes it possible to define a general notion of logic of behavioural constraints. Instances include, for example, start graphs, application and consistency conditions, and temporal logic constraints. We show that the considered semantics can be restricted to a final coalgebra semantics for systems with behavioural constraints.

The paper [GHK97] is concerned with the combination of graph transformation 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 graph transformation language PROGRES is a kind of visual programming language with a well-defined static type concept (defined by Schürr, Winter and Zündorf in '95). Its integrated and incrementally working type checker is able to check the static semantics of a specification at compile time. Rather recently new graphical as well as textual constructs were added which allow the definition of first-order logic integrity constraints. These constraints together with pre- and postconditions of graph transformation rules and programs are checked at runtime [MSW98], thereby offering new means to check the correctness of constructed specifications. Related work at the Technical University of Berlin shows that it is possible to verify the correctness of graph transformations rules with respect to a subset of PROGRES integrity constraints.

4.1.4 Language Design

In [TER] the AGG-language integrating conventional textual programming in Java and graph transformation is presented. AGG may be understood as a hybrid programming language which offers visual as well as textual concepts. It's due to the programmer's decision what part of the application is realized visually and what textually. Combining visual and textual programming in this way, programming should become easier because of visual concepts. On the other side, the wheel is not reinvented again, since well elaborated programming concepts, tools and class libraries mirroring the state of the art of programming can be used.

The (re-)design and improvement of the graph transformation language PROGRES was and is an ongoing activity at the University of Technology in Aachen during the last 10 years. Based on experience reports of the PROGRES language user community we are now busy to simplify and document the language's syntax and to especially to replace unusual syntax constructs by standard notations of the new object-oriented modeling language UML (cf. the PROGRES online documentation accessible via the URL *http://www-i3.informatik.rwth-aachen.de/research/progres*).

4.2 Tool Issues

The APPLIGRAPH subarea *Tool Issues* coordinates the activities of all partners concerning the design and realization of graph transformation tools which support e.g.

- design of concurrent and distributed (software) systems,
- prototyping software and process engineering tools,
- specification of visual languages,
- ...

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 specification and programming tools is unfeasible without the development of appropriate specification and programming languages. And once realized tools have to be applied in order to get any feedback about their usefulness for certain application domains.

One of the major tasks of the tool issue subarea is to assemble and disseminate any available information about graph transformation based tools and their applications. As a first starting point a world wide web page was created which lists all known graph transformation tools and applications (cf. *http://www-i3.informatik.rwth-aachen.de/private/andy/tools.html*). Furthermore, this page contains pointers to information about related topics such as graph drawing tools, visual programming environments, graph-oriented database systems, and the like.

For further details concerning attractive graph transformation tool application areas and experiences in using graph transformation tools the reader is referred to [BS97] and the world wide web page
http://www-i3.informatik.rwth-aachen.de/private/andy/GraphRewriting.

In the following, the tool building activities of all APPLIGRAPH members are listed, divided into the same subtopics as in the working group proposal for APPLIGRAPH.

4.2.1 Editors, Analysers, and Parsers

The Aachen group continued its research activities concerning *coupled graph grammars*. They are mainly used for the a posteriori integration of software engineering tools and chemical process engineering tools. Based on the concepts presented in [Gru97] and refined in [GNS98] an editor and analyser for coupled graph grammars is under development which will be an integrated part of the PROGRES graph transformation environment. This topic is closely related to the development of new graph grammar parsing algorithms. Based on the parsing algorithm published in [RS97] a pair of coupled graph grammars A and B may be used as follows to keep related engineering documents in a consistent state: graph grammar A is used to parse a source document, graph grammar B is then used to generate the related target document.

The programming environment PROGRES itself was extended such that it is now able to handle static integrity constraints for a class of manipulated graphs [MSW98]. Last but not least considerable efforts were necessary to port PROGRES from Sun workstations to Linux PCs, thereby increasing its potential user community considerably (cf. world wide web page
http://www-i3.informatik.rwth-aachen.de/research/progres).

4.2.2 Interpreters, Compilers, and Optimisers

The research group in Antwerp continued its work on the development of a prototype implementation of a compiler/interpreter for an actor language based on graph rewriting. Basically, it has been demonstrated that an actor language with a traditional structure can be automatically translated into a set of graph rewriting production schemes, which are instantiated and applied by the interpreter. Since each message in an actor configuration determines a subgraph to be rewritten, the pattern matching needed for the production applications is trivial, and the concurrent interpretation can in principle be very fast.

A different group of researchers related to the Bremen group presented in [VM97] an algorithm for graph matching. It reduces the problem of finding a homomorphic image of a pattern graph in a target graph to that of finding homomorphic images of every connected component of the pattern in the target. For every connected component, the algorithm performs a combinatorial search bounded by a pruning operator. The algorithm can be applied to directed graphs as well as to undirected graphs, and it can also be specialized to find isomorphic images only.

Finally, the Berlin group continued the development of the graph transformation system AGG (for Algebraic Graph Grammar system). It is an ongoing activity, putting into practice the algebraic Single-Pushout approach with several extensions. Although retaining the name AGG (coined in '93 by Löwe and Beyer), 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 AGG system 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.

The graph matching in AGG is formalized and solved as a constraint satisfaction problem. For a provably correct implementation of a proper graph transformation step, which is given by the categorical construction of a colimit, the realization relies on the colimit library of [Wol98]. Moreover, AGG contains an attribute manager. This component allows to annotate the nodes and arcs of a graph with values of arbitrary Java types, which can then be operated on using arbitrary Java methods in graph rules. With this concept, powerful means are provided for smooth integration of textual and visual programming.

Taentzer, Ermel and Rudolf [TER] give a survey of the concepts, functionality, and use of AGG 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.

4.2.3 Generating and Prototyping Tools

The Aachen group and the scientific APPLIGRAPH correspondents in Paderborn continued their work on using PROGRES for rapid prototyping graphical software engineering tools. Krapp developed in his Ph.D. thesis a new rapid prototyping framework for this purpose. Furthermore, he tested this framework by realizing a quite complex software process modeling tool [Kra98]. Radermacher and Cremer use the same framework for prototyping software reengineering and transformation tools. Finally, Jahnke and Zündorf in Paderborn used the rapid prototyping engine of PROGRES for generating first versions of a bus route tracking system, a distributed course planning and information system, and a tool for migrating relational to object-oriented databases [JZ98, JSZ97].

4.3 Application Domains

4.3.1 Visual Languages

Defining the syntax and semantics of visual (programming) languages by means of graph grammars or graph transformation languages is one of the main research areas of the PROGRES group at the University of Technology Aachen. Syntax-directed research activities together with people from Leiden lead to the development of a new graph grammar parsing algorithm based on so-called Layered Graph Grammars [RS97]. Refined versions of this algorithm were and are successfully used at different sites in Sidney and Bari to define the syntax of visual languages and to generate visual programming environments [ZZ97].

Furthermore, the graph transformation language PROGRES was used to specify the syntax and semantics of the new visual dataflow programming language BDL [Sch97]. Using this specification as input, a rapid prototype of a programming environment for BDL was generated.

The technical report [Bar97] is a summary of an analysis of visual languages starting at the end of 1996. The reason for this analysis is the development of a visual (programming) environment based on algebraic specification techniques and graph grammars. In the context of software engineering this analysis can be understood as an informal requirement specification. It discusses some application areas of visual languages in general and diagram editors in particular. Furthermore, the report treats some approaches with a similar aim, namely the development of visual environments based on graph grammars. It ends with an informal requirement specification corresponding to how to represent a diagram, and some conceptual ideas for tool support.

Based on the ideas mentioned in [Bar97] the concepts for a visual environment are presented in [BT97]. The main and very new feature is the graphical definition of visual languages in general which forms the basis to generate supporting editors. Roughly, a user has to determine the visual means of a visual language and links between these means leading to an algebraic specification. On top of the algebraic specification graph grammar rules are used to define visual languages rules. These rules define the edit commands of a syntax directed editor.

In [DKL97] some criteria are presented that can be used to disprove context-freeness of collage languages. Such criteria are of interest because the usual tools like pumping lemmata fail to be useful in the case of collages. Roughly speaking, it is shown that (a) the number of parts of collages in a context-free collage language grows at most linearly and (b) the volume of parts grows exponentially (if it grows at all).

4.3.2 Database Models and Information Systems

The Unified Modeling Language (UML) is becoming a standard as a general-purpose visual modeling language to specify and visualize software systems. It comprises a number of diagram forms used to describe particular aspect of software artifacts: among them, state diagrams describe behavioral aspects of the system. In [GPP98] we show how to transform UML state diagrams into graphs by making explicit the intended semantics of the diagram. The process of state expansion in nested diagrams is explained by graph transformations in three steps: (1) adding boundary nodes introducing a precise interface for the state to be expanded; (2) expanding the state; and (3) removing the boundary nodes. The rules for introducing and removing boundary nodes are fixed once and for all UML models, while the rule for state expansion is the responsibility of the UML modeler who can ignore, thanks to the other rules, the context in which the expansion will be used.

4.3.3 Concurrent and Distributed Systems

The potential use of graph rewriting as a conceptual tool for the implementation of a concurrent languages, e.g. actor languages, is being further investigated. It became clear that the method proposed has the important advantage that the scheduling of production applications can be done purely on the basis of efficiency considerations, since the graph rewriting systems involved are confluent. An effort is also undertaken to use the expressive power of graph rewriting (as opposed to, e.g. Petri nets) in order to obtain a more satisfactory treatment of timing constraints.

Based on the observation that graphs and graph grammars can be fruitfully used to model distributed systems which behave in a synchronized way, [MR98] propose a model where graphs represent networks of processes, and suitable graph rewriting rules, based on the combination of simple context-free productions via a synchronization mechanisms, represent the dynamics of such systems. They use well-known constraint propagation techniques to cope with the problem of efficiently combining productions together. This helps reducing the possible choices for each process and thus simplify the combination problem. This same approach is also be modelled via the Tile Model [GM98b], which has been used to model a graph rewriting formalism aimed at describing the concurrent, synchronized evolution of local processes. Tiles are much like SOS inference rules, but they can be composed horizontally and vertically to build larger proof steps. Tile systems can be seen as double categories and tiles themselves as double cells.

Pullback rewriting has been introduced as a category theoretical framework for node rewriting in graphs. In [Kle98], it is used to describe refinements in Petri nets and to characterize a particular net refinement technique.

Modeling of distributed systems by distributed graph transformation as introduced by Taentzer in '96 offers a clear and elegant description of dynamic networks, distributed actions as well as communication and synchronization using a graphical notation. Moreover, distributed graph transformation offers the possibility to describe splitting and joining of local graphs as well as parallel transformations in local systems. In [Koc97a] distributed graph transformations are extended by distributed application conditions needed for specifying transaction models for distributed systems suitable. The concepts developed are applied to distributed transactions with two-phase commit.

In [TFKV] the visual design of distributed system by distributed graph transformation is presented. The kernel of this technique focuses 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 serie-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.

In [Koc97b] graph transformations is used to specify transaction models.

ACID transactions are used as an example for this specification and it is shown how the property of isolation of these transactions can be checked by a constructive procedure developed in the field of graph transformations.

[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 one 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. As sample application a distributed configuration management occuring in distributed software dev-elopement is used.

4.3.4 Computer-supported Cooperative Work

It is the main purpose of the "Sonderforschungsbereich IMPROVE" at the Technical University of Aachen (supported by the "Deutsche Forschungsgemeinschaft") to develop new concepts for the a posteriori integration of chemical process engineering tools. One of its central components is an administration system which supports simultaneous as well as concurrent engineering activities. A first rapid prototype of this system generated from its PROGRES specification. It uses dynamic task nets for modeling chemical engineering processes. Task nets may be modeled quite naturally as attributed graphs. Moreover, all operations on these nets, including editing, analysis, and execution, are specified by a programmed graph rewriting system [HKWJ97].

Contributions to the semantic foundation of Computer-supported Cooperative Work are [FM97, BM97a, BM97b]. In [FM97], a tile-based bisimilarity semantics for the asynchronous pi-calculus of Honda and Tokoro is presented, and it is proved equivalent to the ordinary semantics. 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 workflow mechanisms.

In [BM97a, BM97b] "zero-safe nets" are proposed as an extension of ordinary P/T Petri nets. The main feature of zero-safe nets is a primitive notion of transition synchronization. Besides ordinary places, called stable places, zero-safe nets come equipped with zero places, which are empty in any stable marking. A connected transaction is a complex computation, which may use zero tokens as triggers, but defines an evolution between stable markings only. Results relating zero-safe nets and traditional nets via adjunctions are presented for both the collective and the individual token philosophy.

4.3.5 Design and Implementation of Programming Languages

For the lazy functional graph rewriting language CONCURRENT CLEAN, term graph rewriting is used as computational model for both language and implementation. Language concepts like lazy copying (for parallel evaluation) and uniqueness typing (for allowing destructive updates in a pure functional setting) are directly based on the notion of graph rewriting. Efficient implementations try to replace pure graph rewriting by semantically equivalent stack and register based machine instructions. In [Ach97], a library written in CLEAN is presented offering the possibility to define platform independent GUI-interfaces using an object oriented style of programming. The paper [Ser97] proposes a new language primitive allowing efficient multi-casting of graphs to processes.

The operational semantics of languages integrating functional and logic programming are usually based on narrowing which combines rewriting with unification. In practice, narrowing is often implemented on graph-like data structures for efficiency reasons. The completeness of such implementations is studied in [HP97]. Completeness results for three narrowing strategies are established: minimally collapsing narrowing, maximally collapsing narrowing and basic narrowing. The result on basic narrowing improves and corrects a recent result of Krishna Rao (published in *Proc. JICSLP '96*).

4.3.6 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 [EHTE97] and [EEHT97], 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. We define a construction for automatic view integration 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.

In distributed object-oriented systems, middleware products like CORBA are employed to perform the actual distribution on a network of computers. Existing programs use object-based abstractions, but cannot directly be distributed in the form of a CORBA system, e.g. because not all data types have corresponding IDL definitions. We are currently developing a tool which supports the distribution process of an existing program. Such a program is modeled as an attribute graph, the needed program transformations from a monolithic program to the required distributed version are implemented as PROGRES graph transformations [Rad97], [CKNR98].

Furthermore, a tool prototype was developed which supports the migration of legacy applications into a distributed environment. As first step of a migration process we have to reverse engineer an application to get a description on a higher level of abstraction. The acquired information is stored in form of graphs (call graphs, control flow etc.) The necessary structural changes are performed on the reverse engineering graphs with the help of PROGRES graph transformation rules [Cre98].

4.3.7 Term Graph Rewriting

In [AKP97] a survey of confluence properties of (acyclic) term graph rewriting is presented. Results and counterexamples are given for plain applications of rewrite rules, extensions with the operations of collapsing and copying, and with both operations together. Collapsing and copying together constitute bisimilarity of term graphs. These results are extended in [AKP98] by considering term graph rewriting *modulo bisimilarity*. This notion of rewriting turns out to be a proper extension of term rewriting but behaves with respect to termination and confluence exactly like term rewriting.

Computations by term graph rewriting terminate more often than computations by the corresponding term rewrite systems. For this reason, in [Plu97] *simplification orders* on term graphs are introduced and shown to be well-founded. Thus, these orders can be used in termination proofs. A *recursive path order* on term graphs is defined by analogy with the well-known order on terms, and is shown to be a simplification order.

In [CD97] it is shown that the rewriting of cyclic term graphs (which denote rational, possibly infinite terms) is adequate with respect to rational parallel term rewriting. The approach constrasts with the "classical" one in that a single graph reduction corresponds to a single (but possibly infinite) parallel term rewrite step whereas, classically, a single graph reduction may correspond to an infinite sequence of term rewrite steps. A related result is presented in [CG98b], where infinite parallel term rewriting is shown to be an adequate semantics for the rewriting of μ-terms, i.e., terms enriched with an auto-instantiation operator.

A new categorical description for term graph rewriting is proposed in [CG97]. It is already known that term rewriting systems can be faithfully described by a Cartesian 2-category, where horizontal arrows represent terms, and cells represent rewriting sequences. A similar, original 2-categorical presentation is presented for term graph rewriting. In [CG98a] the main result is a categorical characterization of term graphs that parallels the well-known characterization of terms as arrows of the algebraic theory of a given signature. In particular, it is shown that term graphs over a signature Σ are one-to-one with the arrows of the free gs-monodial category generated by Σ. Such a category satisfies all axioms for Cartesian categories but for the naturality of two transformations providing in this way an abstract and clear relationship between terms and term graphs.

4.3.8 Neural Networks

In [FKB98a] graph transformations are introduced as 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.

In [FKB98b] a topology changing training algorithm for Probabilistic Neural Networks, more exactly the Dynamic Decay Adjustment (DDA) algorithm, is specified by graph transformation rules. With the help of concepts developed in the field of graph transformations 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 more generally 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

[Ach97] P.M. Achten. Interactive functional objects in CLEAN. In C. Clack, T. Davie, and K. Hammond, editors, *Implementation of Functional Languages, 9th International Workshop (IFL'97)*, LNCS, 1997.

[AKP97] Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Confluent rewriting of bisimilar term graphs. In *Proc. Fourth Work-*

shop on Expressiveness in Concurrency, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1997.

[AKP98] Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Bisimilarity in term graph rewriting. Report SEN-R9801, CWI, Amsterdam, 1998.

[Bar97] R. Bardohl. Application of graph transformation to visual languages. Technical Report 97-10, Technische Universität Berlin, 1997.

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

[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*, LNCS. Springer Verlag, 1998. To appear.

[BM97a] R. Bruni and U. Montanari. Zero-safe nets, or transition synchronization made simple. In *Proceedings EXPRESS'97*, volume 7 of *ENTCS*. Elsevier, 1997.

[BM97b] R. Bruni and U. Montanari. Zero-safe nets: The individual token approach. In *Proc. 12th WADT Workshop on Algebraic Development Techniques*, volume 1376 of *LNCS*. Springer Verlag, 1997.

[BS97] D. Blostein and A. Schürr. Computing with graphs and graph rewriting. Technical Report 97-8, RWTH Aachen, Fachgruppe Informatik, Ahornstr.55, (D)-52056 Aachen, 1997.

[BT97] R. Bardohl and G. Taentzer. Defining visual languages by algebraic specification techniques and graph grammars. In *Proc. of Theory of Visual Languages '97*, 1997.

[CD97] A. Corradini and F. Drewes. (Cyclic) term graph rewriting is adequate for rational parallel term rewriting. Technical Report TR-97-14, Dipartimento di Informatica, Pisa, 1997.

[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.

[CG98a] A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *APCS*, 1998. To appear.

[CG98b] A. Corradini and G Gadducci. Rational term rewriting. In M. Nivat, editor, *Proceedings FoSSaCS'98*, volume 1378 of *LNCS*. Springer Verlag, 1998.

[CKNR98] K. Cremer, P. Klein, M. Nagl, and A. Radermacher. Prototypische Werkzeuge zur Restrukturierung und Verteilung von Anwendungssystemen. In St. Jähnichen, editor, *Informationstechnik im Zeitalter des Internet, Proc. ONLINE '98, Congress VI*, pages C.630.01–C630.26. ONLINE, 1998.

[Cre98] Katja Cremer. A tool supporting the re-design of legacy applications. In Paolo Nesi and Franz Lehner, editors, *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, pages 142–148. IEEE Computer Society, 1998.

[DE98] Frank Drewes and Joost Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Information and Computation*, 1998. To appear.

[DKL97] Frank Drewes, Hans-Jörg Kreowski, and Denis Lapoire. Criteria to disprove context-freeness of collage languages. In B.S. Chlebus and L. Czaja, editors, *Proc. Fundamentals of Computation Theory XI*, volume 1279 of *Lecture Notes in Computer Science*, 1997.

[Dre97] Frank Drewes. On the generation of trees by hyperedge replacement. In I. Prívara and P. Ružička, editors, *Proc. Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 229–238, 1997.

[Dre98] Frank Drewes. A characterization of the sets of hypertrees generated by hyperedge-replacement graph grammars. *Theory of Computing Systems*, 1998. To appear.

[EEHT97] G. Engels, H. Ehrig, R. Heckel, and G. Taentzer. A combined reference model- and view-based approach to system specification. *Int. Journal of Software and Knowledge Engeneering*, 1997. To appear.

[EEKR98] 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, 1998. In preparation.

[EHTE97] G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A view-oriented approach to system modelling based on graph transformation. In *Proc. of ESEC/FSE'97*. Springer, LNCS 1301, 1997.

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

[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*, 1998. To appear.

[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*, 1998. To appear.

[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*, 1998. To appear.

[FM97] G.L. Ferrari and U. Montanari. A tile-based coordination view of asynchronous pi-calculus. In *Proceedings Mathematical Foundations of Computer Science*, volume 1295 of *LNCS*, pages 52–70. Springer-Verlag, 1997.

[GHK97] Fabio Gadducci, Reiko Heckel, and Manuel Koch. Combining Graph Transformations with Temporal Logic. Technical Report 98-??, Technical University of Berlin, 1997. To appear.

[GM98a] F. Gadducci and U. Montanari. Axioms for contextual net processes. In *Proceedings ICALP'98*, LNCS. Springer Verlag, 1998. To appear.

[GM98b] 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, 1998.

[GNS98] S. Gruner, M. Nagl., and A. Schürr. Fine-grained and structure-oriented integration tools are needed for (product) development processes. Technical Report 98-2, RWTH Aachen, Fachgruppe Informatik, Ahornstr.55 , (D)-52056 Aachen, 1998.

[GPP98] M. Gogolla and F. Parisi-Presicce. State diagrams in UML: A formal semantics using graph transformations. In Manfred Broy, Derek Coleman, Tom S. E. Maibaum, and Bernhard Rumpe, editors, *Proceedings PSMT'98 Workshop on Precise Semantics for Modeling Techniques*. Technische Universität München, TUM-I9803, April 1998. Also in Techn. Rep. 98-07, Universitá di Roma La Sapienza.

[GRPPS97a] M. Große-Rhode, F. Parisi-Presicce, and M. Simeoni. Concrete spatial refinement constructions for graph transformation systems. Technical Report 97-10, Universitá di Roma La Sapienza, 1997.

[GRPPS97b] M. Große-Rhode, F. Parisi-Presicce, and M. Simeoni. Spatial and temporal refinement of typed graph transformation systems. Technical Report 97-11, Universitá di Roma La Sapienza, 1997.

[GRPPS98] M. Große-Rhode, F. Parisi-Presicce, and M. Simeoni. Refinements and modules for typed graph transformation systems. Technical Report 98-2, Universitá di Roma La Sapienza, 1998.

[Gru97] S. Gruner. Werkzeugspezifikation mit Schemakorrespondenzen. *Softwaretechnik-Trends*, 17(3):39–42, 1997.

[HEWC97a] R. Heckel, H. Ehrig, U. Wolter, and A. Corradini. Double-pullback transitions and coalgebraic loose semantics for graph transformation systems. *Applied Categorical Structures*, 1997. Submitted.

[HEWC97b] R. Heckel, H. Ehrig, U. Wolter, and A. Corradini. Integrating the specification techniques of graph transformation and temporal logic. In *Proc. of MFCS'97, Bratislava*, volume 1295 of *LNCS*. Springer Verlag, 1997.

[HEWC97c] R. Heckel, H. Ehrig, U. Wolter, and A. Corradini. Loose semantics and constraints for graph transformation systems. Technical Report 97-07, TU Berlin, 1997. http://www.cs.tu-berlin.de/cs/ifb/TechnBerichteListe.html.

[HP97] Annegret Habel and Detlef Plump. Completeness of narrowing in non-copying implementations. Informatik-Bericht 19/97, Universität Hildesheim, Institut für Informatik, 1997.

[JSZ97] J.-H. Jahnke, W. Schäfer, and A. Zündorf. A design environment for migrating relational to object oriented database systems (abstract). In *Software Engineering and Database Technology, Dagstuhl-Seminar-Report 173, March 17 - 21*, 1997.

[JZ98] J.-H. Jahnke and A. Zündorf. Specification and implementation of a distributed planning and information system for courses based on story driven modelling. In *IEEE Conf. Proc. IWSSD-9, Kyoto*. IEEE Computer Society Press, 1998. To appear.

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

[KKS97] Hans-Jörg Kreowski, Sabine Kuske, and Andy Schürr. Nested graph transformation units. *International Journal of Software Engineering and Knowledge Engineering*, 7(4):479–502, 1997.

[Kle98] Renate Klempien-Hinrichs. Net refinement by pullback rewriting. In M. Nivat, editor, *Proc. Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 189–202. Springer-Verlag, 1998.

[Koc97a] M. Koch. Bedingte verteilte Graphtransformation und ihre Anwendung auf verteilte Transaktionen. Technical Report 97-11, TU Berlin, FB 13, 1997.

[Koc97b] M. Koch. Graphtransformation als universelles Beschreibungs- und Ausführungsmodell für Transaktionen. *GI Rundbrief Datenbanken*, 11/97, May 1997.

[Kra98] C.A. Krapp. *An Adaptable Environment for the Management of Development Processes*. Augustinus, 1998. Doctoral Dissertation, RWTH Aachen.

[MR98] U. Montanari and F. Rossi. Graph rewriting, constraint solving and tiles for coordinating distributed systems. *APCS*, 1998. To appear.

[MSW98] M. Münch, A. Schürr, and A.J. Winter. Integrity constraints in the multi-paradigm language progres. Technical Report DSSE-TR-98-2, Department of Electronics and Computer Science, University of Southampton, 1998.

[Plu97] Detlef Plump. Simplification orders for term graph rewriting. In I. Prívara and P. Ružička, editors, *Proc. Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 458–467, 1997.

[Plu98] Detlef Plump. Termination of graph rewriting is undecidable. *Fundamenta Informaticae*, 1998. To appear.

[Rad97] Ansgar Radermacher. Distribution of Existing Programs via CORBA. In *International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97)*. Chapman & Hall, 1997. To appear.

[RMEK98] 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, 1998. In preparation.

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

[RS97] J. Rekers and A. Schürr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages and Computing*, 8(1):27–55, 1997.

[Sch97] A. Schürr. BDL - a nondeterministic data flow programming language with backtracking. In *Proc. VL'97 13th Int. IEEE Symp. on Visual Languages*, pages 398–405. IEEE Computer Society Press, 1997.

[Ser97] P.R. Serrarens. Using multicasting for optimising data-parallelism. In C. Clack, T. Davie, and K. Hammond, editors, *Implementation of Functional Languages, 9th International Workshop (IFL'97)*, LNCS, 1997.

[SW97] A. Schürr and A.J. Winter. Formal Definition and Refinement of UML's Module/Package Concept. In J. Bosch and S. Mitchell, editors, *Object-Oriented Technology — ECOOP '97 Workshop Reader*, Lecture Notes in Computer Science, pages 211–215. Springer-Verlag, Berlin, 1997.

[SW98] 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*. Physica-Verlag, Heidelberg, 1998.

[Tae97] G. Taentzer. Distributed graphs and graph transformation. *Applied Categorical Structures*, 1997. To appear.

[TER] G. Taentzer, C. Ermel, and M. Rudolf. AGG-approach: Language and tool environment. In [EEKR98].

[TFKV] G. Taentzer, I. Fischer, M Koch, and V. Volle. Visual design of distributed systems by graph transformation. In [RMEK98].

[VM97] Gabriel Valiente and Conrado Martínez. An algorithm for graph pattern-matching. In *Proc. 4th South American Workshop on String Processing*, volume 8 of *International Informatics Series*, pages 180–197. Carleton University Press, 1997.

[Wol98] Dietmar Wolz. *Colimit Library for Graph Transformations and Algebraic Development Techniques.* PhD thesis, Technical University of Berlin, 1998.

[WS97] A.J. Winter and A. Schürr. Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems. Technical Report AIB 97-3, Dept. of Computer Science, RWTH Aachen, 1997.

[ZZ97] D.-Q. Zhang and K. Zhang. Reserved graph grammars: A specification tool for diagrammatic VPLs. In *Proc. VL'97 13th Int. IEEE Symp. on Visual Languages*, pages 284–291. IEEE Computer Society Press, 1997.