Appligraph
Applications of Graph Transformation

ESPRIT Working Group No. 22565

Third Annual Progress Report

Hans-Jörg Kreowski and Detlef Plump
(editors)

May 2000

Contents

1 Introduction

This report summarizes the activities in the third year of the ESPRIT Working Group APPLIGRAPH, covering the period from April 1, 1999, to March 31, 2000. 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* (coordinated by Gregor Engels)
- *Tool Issues* (coordinated by Manfred Nagl)
- *Application Domains* (coordinated by Francesco Parisi-Presicce)

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:

Aachen University of Technology: Manfred Nagl (team leader), Bernhard Westfechtel, André Marburger, Boris Boehlen, Ansgar Schleicher, Dirk Jäger, Oliver Meyer, Felix Gatzemeier, Manfred Münch, Katja Cremer (until May 2000), Anita Behle (until March 2000), Peter Klein (until June 2000), Ansgar Radermacher (until April 1999), Stefan Gruner (until July 1999), Roland Baumann (until April 1999)

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, Claudia Ermel, Thorsten Schultzke, Olga Runge, Maike Gajewsky, Felix Bübl, Michael Matz, Ingo Weinhold, Marta Simeoni, Aliki Tsiolakis, Magnus Niemann

University of Bremen: Otthein Herzog (team leader), Hans-Jörg Kreowski (team leader and Working Group coordinator), Detlef Plump (Working Group manager), Frank Drewes (until April 2000), Sigrid Ewert, Martin Gogolla, Berthold Hoffmann, Gregor Joeris, Renate Klempien-Hinrichs, Peter Knirsch, Sabine Kuske

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

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:[1] Gregor Engels (team leader), Ralph Depke, Reiko Heckel, Jochen Küster, Katharina Mehner, Stefan Sauer, Annika Wagner

University of Pisa: Ugo Montanari (team leader), Paolo Baldan, Stefano Bistarelli, Roberto Bruni, Andrea Corradini, Andrea Maggiolo-Schettini, Emilio Tuosto

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)

[1] University of Paderborn—whose team leader Gregor Engels had moved from Leiden to Paderborn—became a new site of APPLIGRAPH in February 1999

3 Summary of Activities

3.1 Research

During the third 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 the four books [EEKR99, EKMR99, EEKR00, NSM00], the workshop proceedings [ET00], 25 papers published or accepted for publication in journals and books, 71 papers published or accepted for publication in conference and workshop proceedings, one habilitation thesis, 8 doctoral dissertations and 11 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 and Symposium AGTIVE, the Joint APPLIGRAPH/GETGRATS Workshop GRATRA 2000, two Subgroup Meetings and two Team Leader Meetings have been organized during the third year of the Working Group.

International Workshop and Symposium AGTIVE. The International Workshop and Symposium *AGTIVE — Applications of Graph Transformation with Industrial Relevance* took place in Kerkrade (The Netherlands), September 1–3, 1999. This event was organized by M. Nagl, A. Schürr and M. Münch. The scientific programme consisted of 32 talks, a panel discussion and tool demonstrations. There were 51 participants, among them 24 APPLIGRAPH members and 5 Scientific Correspondents of the Working Group.

Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000). This workshop took place in Berlin, March 25–27, 2000, as a satellite event of the ETAPS 2000 conference. The workshop was a joint event with the TMR Research Network GETGRATS, and was organized by H. Ehrig and G. Taentzer. There were 35 scientific talks, a panel discussion and 10 tool demonstrations. The workshop had 76 participants, among them 33 APPLIGRAPH members and 5 Scientific Correspondents of the Working Group.

Subgroup Meeting in Kerkrade. Prior to the AGTIVE Symposium, on August 31, 1999, there was a Subgroup Meeting on *Design of Graph Transformation Tools* in Kerkrade. The meeting had 9 talks which were related to various tool development activities.

Subgroup Meeting in Bremen. A Subgroup Meeting on *Visualization and Verification of Graph Transformation (VV '99)* took place in Bremen, December 9–10, 1999. This meeting was jointly organized with a Focus Area Meeting of the TMR Network GETGRATS. The programme consisted of 7 talks which addressed different aspects of visualization and verification.

Fourth and Fifth Team Leader Meeting. These meetings took place during the AGTIVE Symposium and the GRATRA 2000 Workshop, respectively. Several administrative and financial issues, and the future events of the Working Group were discussed.

3.3 Future Events

The following events are planned for the final year of APPLIGRAPH:

Workshop on Graph Transformation and Visual Modeling Techniques. This event is co-organized by APPLIGRAPH and will take place in Geneva (Switzerland), July 14–15, 2000, as a satellite event of the ICALP 2000 conference.

Subgroup Meeting in Paderborn. This meeting will take place September 5–6, 2000, at the University of Paderborn. Its purpose is to develop a common exchange format for graph transformation.

3.4 Information Dissemination

In the third 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 second and the third volume of the *Handbook of Graph Grammars and Computing by Graph Transformation* appeared with World Scientific Publishers, see [EEKR99, EKMR99]. The second volume is on *Applications, Languages, and Tools* and the third is on *Concurrency, Parallelism, and Distribution.* The first volume on *Foundations* appeared in 1997, see [Roz97].

Selected Papers of TAGT '98. The Working Group supported the editing of a book with selected papers of the Sixth International Workshop on Theory and Application of Graph Transformations (TAGT '98), which appeared in the series *Lecture Notes in Computer Science* of Springer-Verlag, see [EEKR00].

Proceedings of AGTIVE '99. The proceedings of the International Workshop and Symposium on Applications of Graph Transformations with Industrial Relevance (AGTIVE '99) also appeared in *Lecture Notes in Computer Science*, see [NSM00].

Proceedings of GRATRA 2000. The proceedings of the Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000) appeared as a technical report of the Technical University of Berlin, see [ET00] and `http://tfs.cs.tu-berlin.de/gratra2000`. Full versions of selected papers will appear in special issues of the journals *Mathematical Structures in Computer Science* (Cambridge University Press) and *Science of Computer Programming* (Elsevier Publishers).

APPLIGRAPH NEWS. An *Electronic Bulletin Board* is used by APPLIGRAPH members to distribute paper announcements and related news via the Internet. This service is based on the GRAGRA mailing list 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 home page (`http://www.informatik.uni-bremen.de/theorie/appligraph/home.html`) contains information about the project (synopsis, progress reports and proposal) and links to related information (workshops, graph transformation tools, graph grammar bibliography and graph grammar mailing list).

3.5 Further Activities

To establish a sustained promotion of the area of graph transformation, APPLIGRAPH supported the foundation of the Steering Committee of the International Conference on Graph Transformation (ICGT). The members are Michel Bauderon, Andrea Corradini, Hartmut Ehrig (chair), Gregor Engels, Dirk Janssens, Hans-Jörg Kreowski, Ugo Montanari, Manfred Nagl, Francesco Parisi-Presicce, John Pfaltz, Rinus Plasmeijer, Azriel Rosenfeld and Grzegorz Rozenberg.

4 Scientific Results of the Third Year

The following description of the third 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

In this area topics in graph transformation based languages are investigated, like typing, modularity, refinement, parallelism, concurrency, distribution, optimisation, and correctness.

4.1.1 Modularity

In [GPS00], a refinement concept for graph transformation systems is presented. Graph transformation systems are formal models of computational systems, specified by rules that describe the atomic steps of the system. A refinement of a graph transformation system is given by associating with each of its rules a composition of rules of a refining system, that has the same effect as the original rule. The basic operations on graph transformation rules are sequential and parallel composition, corresponding to temporal and spatial refinements respectively. Syntactically refinements are represented by rule expressions that describe how the refining rules shall be composed.

Spatial and temporal refinement relations between typed graph transformation systems have been introduced in preceding papers. 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.

Furthermore, an abstract categorical approach to modularization of graph transformation systems is introduced in [Sim00]. It is initiated for both — Local Action Systems [Jan99] and typed graph transformation systems over the Double Pushout approach. The abstract module concept proposed is inspired by the algebraic specification modules. Modules are defined over a generic category *cat* of graph transformation specifications and morphisms. These abstract modules are called *cat*–modules. A *cat*–module has three components: an import interface, an export interface and a body. The interfaces describe the relations to other *cat*–modules. More precisely, the export interface specifies the features offered by the module to other modules or the environment, the import interface specifies the parts that are required from other modules, and can be used in the body to implement the export interface. Each component is a graph transformation system specification which consists of a set of graph transformation rules that define its basic steps.

The interconnections between the components are modeled by morphisms of graph transformation specifications.

The module concept presented in [Sim00] supports implementation via refinements, i.e each exported rule is associated with a combination of body rules refining (implementing) it, while information hiding is supported both by the use of types (realizing data hiding) and by using refinements (realizing implementation hiding). Each body rule is hidden inside the module, unless the employed refinement associates an export rule with just that body rule. Moreover, guided by the analogous interconnection operations for algebraic specification modules, we define the union, composition and refinement of modules. The composition realizes the use relation, while the refinement realizes a kind of inheritance relation. Compatibility properties between the proposed operations are proven: one of the most important is the compatibility of composition with respect to refinement of modules.

Two instantiations of *cat*-modules are presented in [Sim00]. The first one concerns typed graph transformation systems over the double pushout approach and the second one concerns local action systems [Jan99]: for both frameworks the concrete structure and concrete results are built up that allow the instantiation. These results are an important step towards the application of local action systems (LAS) and related graph transformation systems as a language for the expression and analysis of distributed algorithms, as they enable one to relate in an elegant way GTS descriptions of a system at various levels of abstraction.

The doctoral dissertation [Win99] introduces a module concept for the graph rewriting (transformation) language PROGRES, which is closely related to the package concept of the standardized object-oriented Unified Modeling Language (UML). It supports a variety of software design patterns including the construction of "Abstract Graph Types" and "Updatable Graph Views".

[HEET99a] presents a systematic approach 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 [HEET99a], these questions are answered for five different module concepts for graph transformation systems. The presentation is illustrated by a reference example, being a sample modular specification of resource management in distributed operating systems.

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. 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.

The notion of a transformation unit together with its interleaving semantics is presented and thoroughly studied in [KK00b]. 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. After illustrating the concept of transformation units with an example from the area of VLSI theory, the authors present various operations on transformation units that model operations on binary relations on graphs. Moreover, some normal forms of transformation units are considered that shed some light on how the different components of transformation units are related to each other. 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 [KK99] and [DKKK00] transformation modules are considered that encapsulate sets of transformation units. A transformation module has an export interface, an import interface, and a formal parameter. With the first two, sets of transformation units can be exported and imported while the latter allows to specify operations on graphs in a loose way. The module concept is illustrated with two examples that model the shortest-path algorithms of Floyd and Dijkstra.

In [HHKK00] another 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 compount 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 non-determinism 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 [Kus00a] a selection of concrete classes of control conditions is discussed which seem to provide reasonable expressive power for specifying and programming with transformation units.

A thorough study of the theory of transformation units is presented in the thesis [Kus00b].

If systems are specified by graph transformation, large graphs should be structured in order to be comprehensible. The papers [DHP00a, DHP00b] present an approach for the rule-based transformation of hierarchically structured (hyper)graphs. In these graphs, distinguished hyperedges contain graphs that can be hierarchical again. This framework extends the well-known double-pushout approach from flat to hierarchical graphs. The paper shows how pushouts and pushout complements of hierarchical graphs can be constructed recursively. Moreover, rules are made more expressive by introducing variables which allow to copy and to remove hierarchical subgraphs in a single rule application.

4.1.2 Parallelism and Concurrency

For Local Action Systems (LAS), [Jan99], recent research on concurrency has concentrated on two topics: on the one hand, the algebraic properties of their processes, and on the other hand the relationship between time, execution order and causal order. It has been shown in [JV99] that subprocesses of a LAS process correspond to abstract operations on graphs, and that the set of these operations, equipped with suitable composition operations, satisfies certain algebraic equations. These results are interesting because they provide a step towards compositional reasoning about systems: the subprocesses describe the execution of the various components of a complex systems.

In [JV00] the notion of a LAS process has been extended by imposing a (temporal) order on the local actions, and it is then shown that the concurrent composition of processes, as opposed to the sequential composition, can be defined only if the alphabet of actions satisfies certain algebraic properties, as was the case in [Jan99]. In this way the notion of concurrency is expressed at the level of the alphabet of actions, rather than on the level of a concrete system.

In [BCM00], mimicking Winskel's construction for Petri nets, a concurrent semantics for (*double-pushout*) *DPO graph grammars* has been provided by showing that each graph grammar can be unfolded into an acyclic branching structure, that is itself a (nondeterministic occurrence) graph grammar describing all the possible computations of the original grammar.

Such an unfolding construction can be described as a coreflection between the category of graph grammars and the category of occurrence graph grammars. The result is shown to hold for a suitable subclass of graph grammars, called *semi-weighted graph grammars*. Unfortunately the coreflection does not extend to the whole category of graph grammars: some ideas for solving the problem are suggested.

With the aim of consolidating the foundations of the concurrency theory for graph transformation systems, the thesis [Bal00] extends to this more general setting some fundamental approaches to the semantics coming from Petri net theory. More specifically, focusing on the so-called *double pushout* (DPO) *algebraic approach* to graph rewriting, the thesis provides graph transformation systems with truly concurrent semantics based on (concatenable) processes and on a Winskel's style unfolding construction, as well as with more abstract semantics based on event structures and domains.

The first part of the thesis studies two generalizations of Petri nets, already known in the literature, which reveal a close relationship with graph transformation systems, namely: *contextual nets* (also called nets with read, activator or test arcs) and *inhibitor nets* (or nets with inhibitor arcs). Extending Winskel's seminal work on safe nets, the truly concurrent semantics of contextual nets is given via a chain of coreflections leading from the category of contextual nets to the category of finitary coherent prime algebraic domains. A basic role is played by *asymmetric event structures*, which generalize prime event structures by allowing a non-symmetric conflict relation. The work is then generalized to inhibitor nets, where, due to the non-monotonicity of the enabling, the causal structure of computations is far more complex, and a new, very general, notion of event structure, called *inhibitor event structure*, is needed to faithfully describe them. The second part of the thesis, relying on the conceptual basis drawn in the first part, focuses on graph grammars. Inhibitor event structures turn out to be expressive enough to model graph grammar computations, and the theory developed for contextual and inhibitor nets, comprising the unfolding and the (concatenable) process semantics, can be lifted to graph grammars. The developed semantics is shown to to be consistent also with the classical theory of concurrency for DPO graph grammars relying on *shift-equivalence*.

In the chapter [BCE+99] of the recently published third volume of the Handbook of Graph Grammars, the authors review the theory of concurrency for the DPO approach to typed graph rewriting.

In [GHL99] the concurrent semantics of double-pushout (DPO) graph rewriting, which is classically defined in terms of shift-equivalence classes of graph derivations, is axiomatized via the construction of a free monoidal bi-category. In contrast to a previous attempt based on 2-categories, the use of bi-categories allows to define rewriting on *concrete* graphs. Thus, the problem of composition of isomorphism classes of rewriting sequences is avoided. Moreover, in order to recover the full expressive power of the formalism via a purely algebraic description, the concept of *disconnected rules* is introduced, i.e., rules whose interface graphs are made of disconnected nodes and edges only. It is proved that, under reasonable assumptions, rewriting via disconnected rules enjoys similar concurrency properties like in the classical approach.

Reactive systems perform their tasks through interaction with their users or with other systems (as parts of a bigger system). An essential requirement for modeling such systems is the ability to express this kind of interaction. Classical rule-based approaches like Petri nets and graph transformation are not suited for this purpose because they assume to have complete control about the state and its transformation.

Therefore, in [EHL$^+$00] a general framework is proposed which extends a given rule-based approach by a loose semantics where the rules of the system (e.g., graph productions or Petri net transitions) are considered as incomplete descriptions of the transformations to be performed: they still determine the changes to the matched substructure but for the remaining part (the context) unspecified changes are possible representing the application of (unknown) rules from the environment. The framework is applied to graph transformation systems in the double-pushout approach as well as place-transition Petri nets leading, respectively, to the concepts of *graph transition* and *open step.*

In [HLEO00], the basics of a theory of concurrency for graph transitions are developed. In particular, a Local Church-Rosser Theorem and a weaker version of the Parallelism Theorem is proved. Conceptually, this allows us to represent both intra-concurrency (within the specified subsystem) and inter-concurrency (between subsystem and environment).

4.1.3 Optimization and Correctness

In [GHK00], graph-interpreted temporal logic is an extension of propositional temporal logic for specifying *graph transition systems* (i.e., transition systems whose states are graphs). Recently, this logic was used for the specification and compositional verification of safety and liveness properties of rule-based graph transformation systems. However, no calculus or decision procedure for this logic was provided, which is the purpose of [GHK00]. First it is shown that any sound and complete deduction calculus for propositional temporal logic is also sound and complete when interpreted on graph transition systems, that is, they have the same discriminating power like general transition systems. Then, structural properties of the state graphs are expressed by *graphical constraints* which interpret the propositional variables in the temporal formulas. For any such interpretation we construct a graph transition system which is *typical* and *fully abstract*. Typical here means that the constructed system satisfies a temporal formula if and only if the formula is true for all transition systems with this interpretation. Fully abstract means that any two states of the system that can not be distinguished by graphical constraints are equal. Thus, for a finite set of constraints one ends up with a finite state transition system which is suitable for model checking.

In [HMP00a], three variants of the double-pushout approach to graph transformation are obtained by considering injective matching and/or arbitrary right-hand morphisms in rules. It is shown that, compared with the traditional approach with arbitrary matching, injective matching provides additional expressiveness for both generating graph languages by grammars without nonterminals and for computing graph functions by convergent graph transformation systems. Moreover, for the most general variant with injective matching and arbitrary right-hand morphisms, sequential and parallel commutativity is established by strengthening sequential and parallel independence. Additionally, in [HMP99, HMP00b], the classical theorems on parallelism and concurrency are adapted to the setting with injective matching.

Context-free hypergraph grammars allow to define sets of hypergraphs in a recursive way. In the literature, three main approaches can be found: hyperedge rewriting (HR), separated handle rewriting (S-HH), and confluent node rewriting (C-hNCE). With respect to their hypergraph-generating power, HR and S-HH grammars are known to be incomparable. In [Kle00a], it is shown that the hypergraph-generating power of remote-free C-hNCE grammars, which form a large natural subclass of C-hNCE grammars, includes properly that of HR and S-HH grammars together. This indicates that confluent node rewriting plays as important a role in generating sets of hypergraphs as it does in generating sets of graphs. Moreover, it is shown in [Kle00b] that for remote-free C-hNCE grammars, the maximal rank of hyperedges indispensable to generate some set of hypergraphs equals the maximal rank of the hyperedges occurring in the hypergraphs of that set. This implies that if such a C-hNCE grammar generates a set of graphs, then it has a C-edNCE normal form.

Updated reference: The contents of the paper [JK00] has been described in the Second Interim Report.

4.1.4 Language Design

One major problem for the specification and verification of software architectures and specially with distributed systems, is when system evolution includes dynamic changes and reconfigurations of components and connections. Paper [HM99] presents a method for specifying reconfigurations or transformations over the topology of the architecture style, being sure that if the transformation can be specified, then its application over the system will be consistent with respect to the expected architecture style configuration. Styles are described by context-free hyperedge graph grammars. In this context, an instance of an architecture style is determined by a graph generated by the grammar.

The formalization of the method is introduced in two ways. The first approach is a visual presentation and has the intention of showing that it can be used by software architects in real life in a simple way. For this, graph grammars represent styles and their generated graphs specific system architectures. A transformation is visualized as a rule that takes a graph generated by the grammar and applies a reconfiguration to part of that graph, obtaining a new graph that is another valid graph of the grammar. The second approach presents a formal model for the first one using lambda-terms and tile sequents showing also that it is implementable. A related approach is proposed in [HM00], where graphs, productions and derivations are characterized as certain terms of a simply typed λ-calculus, and hyperedge replacement can be efficiently implemented as β-conversion.

With PROGRES, two successful principles are combined 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 graph grammar handbook chapter [SWZ97] documents the development of the PROGRES language and its accompanying programming environment.

Based on the module concept introduced in the PhD thesis [Win99] and the research results of the evaluation of this concept, currently a “true” object-oriented extension extension of PROGRES is being developed. This means that graph transformations are added to node class definitions as commonly known from other OO languages such as e.g. C++. This need has shown up pretty quickly after having studied an important application domain of PROGRES, visual programming and modeling languages (see [MS99]).

In [BP00], graph patterns are introduced, adapting a similar notion from string languages to graph languages. Graph patterns are motivated by the need to impose constraints on the overall structure of several graph parts which are required to be isomorphic. After defining formally when a graph satisfies a pattern, it is shown in [BP00] that satisfaction is decidable and a construction is given for a graph grammar to generate all graphs satisfying a given pattern.

In [Gru00b], a meta-typing approach as known for PROGRES is presented for single-pushout graph transformation (SPO). In a first step the node meta-typing system of PROGRES is generalized such that edges may be subject of meta-typing, too. In a second step, it is shown in [Gru00b] how any meta-typed graph grammar can be transformed into equivalent ordinary typed SPO graph grammars.

In [BEMW00], a new approach for introducing a hierarchical graph model is presented. The basic idea is to add a hierarchy concept to an existing graph grammar approach, and thus to separate the hierarchy concept from the underlying graph model. Therefore, this approach is a step towards the definition of a general-purpose hierarchical graph data model.

In [DHK00b], typed graph transformation systems are used to support an integrated modeling of object- and agent-based systems. As requirement specification, global graph transformation rules determine the overall effect of the interaction among agents and objects while abstracting from the communication involved. On the design level they describe local autonomous operations by which agents may react to changes in their environment. Graph transformation rules are combined to story diagrams in order to describe the temporal ordering of an agent's local operations. In [DHK00a], a concept of roles is introduced for a more fine-grained modeling of objects' and agents' structure and behavior.

4.2 Tool Issues

The APPLIGRAPH subarea *tool issues* coordinates 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 specification and programming tools is impossible 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.

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 we made the decision to present each project in a separate subsection, instead of presenting each category of tools in a separate subsection.

4.2.1 The AGG Tool Set

The Attributed Graph Grammar system AGG provides a visual programming environment which is based on graph transformation and Java. The idea is to program the topmost object structures of an application as well as its dynamics visually. For this purpose in [TER99], these structures are designed as graphs which may be attributed by Java objects and further arbitrary data. The application's behavior is described by graph rules using an if-then-programming style. Application of a graph rule changes the structure graph.

Basic data types as well as object classes already available in class libraries may be used to program the application "in the small". Moreover, new classes may be included. This part of the application is done textually using the modern object-oriented programming language Java. The visually and the textually programmed application parts are combined by attributing the structure graphs by attribute values. These may be data as well as object references. The graph rules may be attributed by Java expressions which are evaluated during rule applications. It's up to the programmer's decision how much of the application is realized visually and how much textually.

Combining visual and textual programming in this way, programming should become easier because of visual concepts. On the other hand, 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 tool environment provides graphical editors for graphs and rules as well as a textual editor for Java expressions integrated into the visual editors. Moreover, visual interpretation and debugging of AGG programs is supported.

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 that supports the user who has to write diagram editors. Diagrams are internally modeled by hypergraphs; hypergraph grammars are therefore a suitable means for describing the syntax of a diagram class. *DiaGen* comes with a generator which is used to create an editor based on a formal specification that mainly describes diagram syntax by this hypergraph grammar and editor behavior by graph transformation.

Work on *DiaGen* has been continued into three directions during the last year:

- Automatic layout mechanism
- Combination of free-hand editing and syntax-directed editing
- Complete realization of *DiaGen* with Java.

Editors which have been specified and generated with *DiaGen* now come with an *automatic layout mechanism.* Layout is specified in terms of constraints on the diagram components' attributes like their position on the drawing canvas or their size. These constraints are attached to productions of the hypergraph grammar which specifies the syntax of the diagram class. A derivation of the diagram (resp. of its internal hypergraph model) therefore implies a system of constraints that specifies valid layouts of the diagram. A constraint solver is used to solve the corresponding constraint satisfaction problem and to determine the diagram layout.

Automatic layout is needed when the user modifies some diagram components which requires other components to be modified as well in order not to change the meaning of the diagram. Furthermore, automatic layout can take care of "beautifying" diagrams which have been created by the user sloppily [CMP99].

Automatic layout is also needed when providing *syntax-directed editing.* Previously, editors which have been created with *DiaGen* have supported free-hand editing only. This means that the user can arbitrarily create and change diagrams like with a drawing tool. A hypergraph parser, which is based on the specified hypergraph grammar, then analyzes the internal hypergraph of the diagram, creates structural information about the diagram and detects errors, whereas syntax-directed editing offers a set of distinct editing operations that are geared towards modifications to the meaning of the diagram rather than to the appearance of the diagram as it is the case with free-hand editing. Syntax-directed editing operations modify the internal model of a diagram, and its appearance has to be changed accordingly. This requires a mechanism which updates the layout of the diagram automatically. Syntax-directed editing operations are specified by hypergraph transformations which are applied to the internal hypergraph of the diagram [KM00].

DiaGen is now completely implemented in Java 2. It consists of a framework for diagram editors and a program generator. The editor developer has to provide a textual specification of the diagram language which mainly consists of a hypergraph grammar, layout constraints, and hypergraph transformations which specify syntax-directed editing operations. Hypergraph grammars have to be context-free, or they have to be context-free with embedding which adds some context-sensitive aspects to context-free hypergraph grammars [BMST99]. *DiaGen* uses external constraint solvers. Currently, QOCA [MCF98] (Monash University, Australia) and PARCON [Gri96] (University of Paderborn, Germany) are used. But *DiaGen* also supports diagram layout modules which have been programmed "by hand".

The editor developer still has to program some parts of the diagram editor "by hand". This code has to deal primarily with the visual representation of diagram components which typically makes heavy use of the Java API. The program generator transforms the textual specification into Java classes which are then combined with the "hand-written" classes to a software component which complies with the *Java Bean* standard. Therefore, diagram editors which have been created with *DiaGen* can be used immediately in RAD tools like *JBuilder* (Borland/Inprise) or *Symantec Cafe* (Symantec).

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 [JZ98]. 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 [NZ00]. 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.

4.2.4 The PROGRES Tool Set

The PROGRES tool set combines two successful principles in one formalism: rulse 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 [Mün00]. Its accompanying programming environment comprises a syntax-directed editor, an analysing tool, an interpreter for debugging graph transformations, and a compiler for supporting 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. PROGRES is used in projects with industrial partners such as Ericsson, GEZ (German Television Licensing Council), Aachen-Münchener Insurance Group etc.

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 syntactical and semantical 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, analysis and execution of 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.

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 [Rad00] 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 reverse 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, [Gru99] 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. The project context of chemical engineering supported by graph-grammatically specified software tools is presented in [CNG99].

4.2.5 GenGEd - Generating Graphical EDitors

The GenGEd environment [Bar99, BNS00] supports the visual specification of visual languages. These language specifications are the input of a generic graphic editor that allows for the syntax-aided editing of visual sentences. According to the constituents of a visual language specification, namely an alphabet and a grammar, GenGEd comes up with an alphabet editor and a grammar editor. The alphabet editor supports the drawing of symbols and the definition of connections between symbols. Graphical constraints are used to connect graphical primitives in order to establish one symbol graphic as well as to define the connections between the symbol graphics. In the grammar editor the user is able to define visual language grammars on top of specific alphabets. Such grammars consist of a start sentence and a set of context-sensitive rules. The rules may be enhanced with negative application conditions expressing some semantical features of a specific visual language. The transformation of visual sentences in the grammar editor as well as in the graphic editor for drawing visual sentences is supported by the graph transformation system Agg and the graphical constraint solver ParCon.

4.2.6 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 [Dre98a, Dre98b] (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 massivily parallel implementation of the actor language is possible, based on the concurrent application of actor grammar production.

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 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

A comprehensive and state-of-the-art overview on various applications of graph transformation is presented in [EEKR99]. A number of papers mentioned in this section are contributions to this book. It is structured into seven parts which reflect the seven main current application areas. These are term rewriting and functional languages, visual and object-oriented languages, applications to software engineering, applications to engineering disciplines, application to pictures, implemented specification languages and tools and structuring and modularization concepts. This volume is accompanied by a CD-ROM with (prototype) environments and tools, discussed in various contributions.

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 [BTMS99], 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 carefully considered w.r.t. efficiency of tools, e.g., when parsing visual languages. Context-free as well as different kinds of context-sensitive languages are considered in [BTMS99]. 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.

The overall aim of [Bar99] is the visual definition of visual languages which can be used then to generate a visual editor for the defined language. Whenever the visual means of languages has to be changed, a time and cost intensive reimplementation will be necessary. This can be avoided if generators for visual environments are available. [Bar99] heads towards this goal by presenting GenGEd. This visual environment supports the visual definition of visual languages and the resulting visual language specification is the parameter of a visual language editor for the manipulation of visual sentences. Visual languages definable by GenGEd are not restricted to visual modeling and programming languages. The goal of this environment is to support numerous kinds of visual languages. These can be diagrammatic, iconic or form-based languages used for software modeling or specification as well as other purposes, i.e. teaching children. The presentation in [Bar99] concentrates on visual syntax rather than semantics of visual languages. However, it distinguishes between abstract and concrete syntax. This distinction is a first step towards an extension concerning semantical definitions based on abstract syntax. The visual language specification that can be defined by using GenGEd consists of an alphabet and a grammar. The alphabet is represented by an algebraic graph signature, the grammar is represented by an algebraic graph grammar. Moreover, the productions of the grammar are used as edit commands of the visual language editor supporting syntax-directed manipulation of sentences.

While [Bar99] gives an overview on the whole GenGEd approach, [BNS00] focuses on the visual tool environment. GenGEd offers a hybrid language for defining the syntax of visual languages. So far, the main components of GenGEd are given by an alphabet and a grammar editor. The syntax description is the input of a diagram editor allowing the syntax-directed manipulation of diagrams. Furthermore, [BE00] presents the conceptual model of GenGEd which is based on algebraic graph transformation and algebraic specification techniques. Starting with a type signature for visual alphabets, the user of GenGEd can define a specific VL-alphabet including graphical constraints. The VL-grammar definable on the VL-alphabet consists of a start sentence and a set of VL-rules. Each VL-rule is defined by a graph grammar rule in the sense of algebraic graph transformation.

Visual graph and net based modeling techniques are of growing interest for software specification and development. In [BER00], the GenGEd approach is reviewed and extended in order to support the description of dynamic behavior and animation of Petri net based models. The basic idea is to define visual behavior and animation rules over the rules defining a corresponding Petri net language and to allow a domain specific layout for the animation view of a Petri net model. The well-known specification of *The Dining Philosophers* serves as running example, where the system view is given as a Petri net, and the animation view visualizes directly the behavior of philosophers sitting at a table.

In [Dre99] a tree-based method for the generation of languages of fractals is introduced. This combines the nondeterministic nature of grammars as known from formal language theory with the infinite refinement of pictures studied in fractal geometry. A short introduction to the system Treebag, which allows to generate such languages (among others) is given in [DK00].

In [DKK00], the notions of context-sensitive and TOL collage grammars are introduced as generalizations of context-free collage grammars. Both kinds of picture-generating devices are more powerful than the context-free case. Nevertheless, the size of collages in a TOL collage language can be shown to grow at most exponentially. In contrast to this, there are no such bounds for context-sensitive collage languages because suitable pictorial representations of recursively enumerable sets of strings can be generated. On the other hand, it is still a conjecture that TOL collage languages exist which are not context-sensitive.

In [KKT99], mutually recursive function systems, picture-generating devices known in the area of fractal geometry, are translated into TOL collage grammars. The translation works in such a way that the infinite sequence of pictures which is specified by a mutually recursive function system through the Hutchinson operator contains exactly those pictures which belong to the language generated by the corresponding TOL collage grammar. In other words, the translation is correct.

[BPS00] presents an algorithmic method to obtain pure rewriting systems from logical descriptions of languages. The method, presented in an SPO framework, is applicable to visual language specification. The discussion is restricted to formulae which describe a limited but significant set of local and global properties of hypergraphs. This work has to be seen in the context of constraint-based definition of languages, a field of visual languages, which has led to so-called "unification-based" grammars as well as methods to translate static constraints on sentences into dynamic constraints on their interactive construction.

Updates: [Dre00b] (Technical Report [Dre98d] in 2nd Progress Report) has been accepted for publication in TCS and [Dre00a] has now appeared.

4.3.2 Concurrent and Distributed Systems

Due to the special requirements of distributed systems, it is important that modeling techniques for this kind of systems offer a stringent module concept. Each module has to support the encapsulation of data structure as well as functionality also at runtime. Modular graph transformation, presented in [GPST99] at the example of a distributed file manager, supports these features. Modules are built up of specifications where attributed graphs describe the static data structures, whereas the dynamic behavior is modeled by the controlled application of graph rules. Rule expressions are used to formulate the control flow. Within one module, a (weak) preservation of export and import behavior wrt. the local behavior in the module's body is stated in the sense that an interface derivation is subsumed by a local derivation if it can be performed. Modules may use each other meaning that each import interface has to be connected with an export interface in a way that the import behavior is subsumed by the export behavior.

The previous work has been presented in the double pushout approach to distributed graph transformation which was introduced to model distributed systems both on the topological and the local state level by a graphical and intuitive formalism. [Koc00b] introduces the single pushout approach to distributed graph transformation, that maintains the graphical and intuitive way to model both levels. It compares briefly the double pushout approach with the single pushout approach to sketch the distributed system applications, where the single pushout approach seems to be more suitable.

To develop distributed systems and to ensure their quality and correctness, is very challenging. [Koc00a] introduces distributed graph transformation based on the single-pushout approach as a formal specification technique for the compositional specification of distributed systems. Distributed graph transformations are an intuitive, graphical and rule-based formalism to model the topology and the local data aspects as well as the dynamic behavior of the system. The approach is compositional in the sense that local components are specified locally and are composed by means of synchronization constraints. The operational semantics of the distributed system is composed from the operational semantics of its components.

In [BPS99], an intermediate language is presented that represents transitions defining behaviours of autonomous agents. The language allows a uniform representation of several diagrammatic languages for specification of reactive systems, based on an underlying notion of transition. The translation of graph transformations to this language opens the opportunity for a notion of communication between agents represented by graphs.

The semantics for Local Action Systems introduced in [Jan99] is based on the notion of a LAS process; such a a process is a very simple structure essentally consisting of a DAG with labeled edges. In contrast to process notions for, e.g., Petri nets the definition of a process does not require the specification of a particular system. Thus a problem specification can be given in terms of LAS processes, independently of the possible implementations, and one may then try to prove a proposed implementation correct w.r.t. that specification. It has been investigated to what extent this approach can be used for the verification of distributed algorithms; in [JV99] an example is given of a distributed shortest path algorithm and its proof. The proof uses a number of results about the way subprocesses of a process may be identified with operations on the graphs on the nodes of the process: a number of algebraic equations is shown to hold for those operations.

In [KK00a] it is shown that a simple agent system can be modeled by applying graph transformation. Similar to transformation units as discussed in [AEH$^+$99], agents have local sets of graph rules determining their capabilities and a local control that coordinates the order the rules are applied in (cf. [Kus00a]). All agents of the agent system change a common environment. Based on this notion, it is shown that attributes of agents as well as of agent systems can sensibly be defined within this framework.

In [TFKV99] 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 a reference application is used 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 [FKT00]. 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 but must not. 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.

Also, [Koc99] provides a methodology for the specification of distributed systems based on graph transformation. Major aims are the compositional modeling of the behavior of distributed systems, the specification of distributed system properties and the support of their (automatic) verification. To reach this goal, an integrated approach of two formal methods is proposed, namely *distributed graph transformation* and *temporal logic*. Distributed graph transformation is intended for modeling the behavior of distributed systems, whereas distributed system properties are specified by temporal logic. The combined approach benefits from the strength of each single approach and provides a suitable methodology to model formally the aspects of distributed systems. The approach permits a variety of specification techniques for local data states, where the requirements for the specification techniques are investigated that can be used for the local data. A main contribution of this approach to distributed graph transformation is the compositional operational semantics. This semantic is based on transformation systems introduced by Große-Rhode. The single-pushout approach to distributed graph transformation is integrated with a propositional temporal logic. The integration takes place over a pre-order relation over distributed graphs and results in graph-interpreted temporal formulas and temporal graph models. Graph-interpreted formulas provide a methodology to specify temporal properties graphically, in the same way as the distributed system itself is specified. The integration of distributed graph transformation and temporal logic allows to make use of the variety of verification concepts in the area of temporal logic to check graph-interpreted temporal formulas with respect to a temporal graph model. This work focuses especially on the concept of model-checking for the automatic verification of graph-interpreted temporal formulas. The construction of a typical graph model is provided that collapses a possibly infinite temporal model to a finite one and that supports the reasoning from the local satisfaction of local formulas in a sub-system of the distributed system to global satisfaction of global formulas in the complete distributed system. For illustration all the presented concepts and ideas are applied to a distributed version management system.

In [GMT00], the application of distributed graph transformation to the problem of specifying dynamic change in distributed systems is considered. Change in distributed system 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 which need to know about the change. The other aspect is changing the structure of the distributed system itself. This implies e.g. to add and/or remove a local node or an entire subsystem to/from the distributed system. In some important application areas such operations must be done during runtime without disturbing the unmodified rest of the distributed computing system. In [GMT00], first an overview of the model of change is given and it is shown how the two aspects of change interact exactly. Distributed graph transformation is the technique to realize the change model. An example - a ring database - then shows how the 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.

[TGM99] focuses on the consistency management of multiple viewpoints in typical multi perspectives/stakeholders settings. The key feature realizing such dynamic modifications and extensions is a clean separation between internal operations or actions on an individual node or agent and their structural coordination.

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 [GM00]. 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.

Thanks to the abstract notion of both system configuration and effect, tile logic can find application in the specification of *open ended systems* — where states can be reconfigured at run-time — simply allowing contexts as observations. Note that this can be done also in the presence of structural axioms on the algebra of configurations. This can have many advantages with respect to most SOS formats designed for ensuring that bisimilarity is a congruence. In fact such formats often fail in the presence of structural axioms, and it is therefore necessary either to resort to the largest congruence included in the bisimilarity by an operation of closure 'for all contexts' (which may result in a congruence which is no longer a bisimulation) or to consider *dynamic bisimulation* that performs the context closure during the bisimulation game (recovering the coarsest congruence which is also a bisimulation). This issue is investigated in [BMS00], where it is shown that the compositional framework offered by tile logic is suitable to deal with structural axioms on the algebra of configurations and open ended system specifications, allowing for a finitary presentation of context closure.

The theoretical foundation for a class of simple tile models, called *zero-safe nets*, has been investigated in [BM00b], together with the application to several case studies. In zero-safe nets, both system configurations and effects are represented as multisets of basic elements. They extend Petri nets with the notion of *transaction* — atomic computation obtained via the synchronization mechanism —and can provide a compositional semantic framework for many CCS-like calculi. A tutorial introduction to the subject has also appeared in [BM99b]. In [BM00a] a distributed interpreter for zero-safe nets is defined, which is reminiscent of the ordinary net unfolding.

Finally, in [BM99a] 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 name passing and higher-order features. The case study of the π-calculus, where the tile framework straightforwardly handles name passing and creation, is studied in detail.

4.3.4 Design and Implementation of Programming Languages

The research group in Nijmegen investigates the suitability of the functional language CONCURRENT CLEAN for the timely development and maintanance of quality software. The CLEAN system runs on small machines and generates programs that run relatively fast. One special feature of the language 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.

One major implementation effort is the development of the *Clean system*. This system consists of the Clean compiler and an integrated programming environment, the *Clean IDE*. The Clean IDE controls a suite of tools to create reliable Clean projects, amongst other *profiling* tools, and *foreign language interfacing* tools. The concrete results of the research that is conducted by the group will eventually be integrated in this environment. Examples are *dynamic linking*, *dynamic typing* [Pil99], and a *proof tool* [dMvE99, dMvE00].

To make this project more feasible, and to reduce future maintainance costs, we have been recoding the major part of the Clean compiler in Clean itself. In addition, new language features have been added to the Clean language. This will result in language version 2.0. The current state of this project is that most of the libraries that have been developed by our group can be compiled with the new system. Since this is a rather large project, it is also a good test case for the quality of the generated code. A number of optimisations have been included ([vG99]) to generate even better code.

The Clean IDE has also been rewritten, now using the latest release of the *Object I/O* library [AW00] that allows one to construct programs with Graphical User Interfaces. This library makes good use of the uniqueness type system of Clean to interface with the imperative world. One distinct feature of this library is that interactive objects can have *local state*. The implementation applies the lazy evaluation scheme of Clean [AP99]. In addition, we have investigated how one can prove properties of interactive programs that are constructed with the Object I/O library [HAKP99a, HAKP99b]. One final lead related to the Object I/O library is the research done on its integration with a specialized *game library* for 2D platform games [WPA99].

Besides having a high-quality compiler and programming environment, the success of a programming language is highly dependent on the availability of supporting libraries. In this report year we have released a number of libraries to make Clean useful not only inside the academic environment: *ArgEnv, CLAS, Directory, htoclean, MersenneTwister, Object I/O, 2D Platform Games, TCP, WrapDebug.*

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

4.3.5 Specification of Software Engineering Models

The modeling of behavior in object-oriented systems usually relies on control flow diagrams and (extended) state automatas. These basic techniques do not well support the design in a declarative style. This style is useful for modeling integrity constraints, especially active constraints, on the data model, to perform event handling, to derived new attribute values and associations and to model strategies in business and engineering. In [Tae99], rules are added as a declarative modeling technique for behavior. Roughly speaking, a rule consists of a condition and an action. If the condition is true, the action is performed. The concept of rules is integrated into an existing object-oriented modeling technique by formulating rules on object diagrams. A rule application may test and change a certain object structure. The application may be triggered automatically by a relevant change of this object structure, periodically or by an event occuring in some behavior diagram. This rule concept has an underlying formal semantics on the basis of graph transformation which may be exploited for analyzing a designed rule set concerning e.g. conflicts, dependencies, etc. The graphical notation of the rules follows the UML-notations as far as possible.

The idea of a combined reference model- and view-based specification approach has been proposed recently in the software engineering community. In [HEET99b], 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.

In a model based software development process different kinds of modelling techniques are used for the specification of different aspects of systems. Thereby mostly visual techniques like UML diagrams are used, due to their comprehensibility, clarity, and wide acceptance. In order to obtain a consistent complete specification of the whole system the different models have to be integrated, i.e., conceptually they have to be considered as one specification. For that purpose comparisons, consistency checks, and integrations for models or specifications of different kinds of techniques are needed. In [GR00] a formal semantics approach to the integration of UML diagrams and other (visual) specification techniques is discussed that is based on a reference model as common semantic domain for all specifications. The reference model is formalized by algebra transformation systems which are a generalization of graph transformation systems.

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. Within this changing world living with inconsistencies is natural. 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. [GMT99] demonstrates how inconsistencies management can be used as a tool for requirement analysis by presenting a sample integration of architecture design and performance evolution views. Based on these concepts, a tool environment under development supporting ViewPoint-oriented software development is presented in [GEMT00] and [GMET00]. The environment is called ViewPoint tool and comprises a ViewPoint manager, a template editor and a ViewPoint editor. While the ViewPoint manager serves as a central coordinater for all the activities in the framework, the template editor allows to design a ViewPoint template. The ViewPoint editor is then used to specify in a certain ViewPoint. Furthermore, the tool supports inconsistency mangement. A non-trivial example integrating software architecture and performance evaluation views is shown.

In [Gru00a], another approach to distributed software development based on graph grammars is investigated. A formal framework is given which is able to to uniformly describe distributed modeling tasks as well as re-engineering tasks. This approach which is based on coupling graph grammars has been applied to distributed modeling in chemical engineering. Tool support for this approach has been provided on the basis of PROGRES.

Consistency between different views is also considered in [TE00] where the relation between UML class and sequence diagrams is investigated. Until now, there is no formal approach to handle this issue in the UML report, and this kind of consistency analysis is not supported by UML modelling tools. Both of these problems are addressed in [TE00]. For this purpose, UML class diagrams are represented by attributed type graphs together with graphical constraints and UML sequence diagrams by attributed graph grammars. In this framework consistency of the UML diagrams means that all graphs generated by the attributed graph grammar are typed over the type graph and satisfy all the graphical constraints corresponding to the UML class diagram. Consistency checking techniques of the algebraic theory of graph grammars are used to develop an algorithm which allows to analyze and check this kind of consistency. This algorithm has been implemented in Java using the AGG system independently of existing UML modeling tools.

The general idea of high-level replacement systems is to generalize the concept of graph transformation systems and graph grammars from graphs to all kinds of structures which are of interest in Computer Science and Mathematics. Within the algebraic approach of graph transformation this is possible by replacing graphs, graph morphisms, and pushouts (gluing) of graphs by objects, morphisms, and pushouts in a suitable category. Of special interest are categories for all kinds of labelled and typed graphs, hypergraphs, algebraic specifications and Petri nets.

In [EGPP99], it is shown how some basic results for graph transformation systems in the algebraic double pushout approach can be reformulated in the framework of high-level replacement systems. The specific choice of results concerning local Church-Rosser properties and horizontal structuring is motivated by the results needed in our application areas studied in this contribution. In order to show the great variety of the high-level replacement approach we do not consider specific graphs and graph transformation but algebraic specifications and Petri nets as application domains, where transformation corresponds to rule-based changes of the structure of specifications and nets, respectively.

The first application shows how high-level replacement systems can be instantiated by algebraic specifications. An algebraic transformation rule corresponds to the interface part of an algebraic module specification for software systems. This allows applying high-level replacement techniques to software system design. As an application it is shown how to reuse an algebraic module specification of an airport schedule for the design of a book library.

The second main application shows how rule-based modification of Petri nets can be considered as a special case of high-level replacement techniques. An important result is the compatibility of horizontal structuring of nets with rule-based modification. This result is essential within a case study of a medical information system where the functional essence is developed by rule-based modification from the actual state of the system represented by algebraic high-level nets.

Existing software configuration management systems embody a wide variety of policies for how artifacts can evolve. New policies continue to be introduced. Without a clean separation of configuration management policies from configuration management mechanisms, it is difficult to understand the policies as well as difficult to reason about how they relate. In [PW00], a formal foundation for specifying configuration management policies by viewing the policies in terms of graph transformation systems. Parisi-Presicce and Wolf are not only able to precisely capture the semantics of individual policies, but also to describe formal properties of the relationship between policies the first time.

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 are its abilities 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 meets these requirements.

A further development of DYNAMITE is described in [WJS00]. The AHEAD system addresses the challenges of managing development processes in different engineering disciplines by providing an integrated environment for modeling and managing these processes. Products, activities, and resources are managed in an integrated way; furthermore, AHEAD supports evolving development processes by seamless interleaving of planning and execution. AHEAD is based on programmed graph transformations; tools are generated from a graph-based specification. Finally, a wide-spread object-oriented modeling language (UML) is employed for acquiring process knowledge from domain experts.

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 transformation we are all looking for. The paper [BGM99] discusses the perspectives of applying the graph transformation approach of PROGRES [Mün00] 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).

In [Cre99] and [Cre00] another project is described which is using PROGRES. Cremer describes an approach for reverse engineering existing applications with the aim to migrate an existing monolithic application to a distributed environment. This is done in two steps: First a COBOL85 application has to be reverse engineered. The result of that step is a cleanly structured distributable application. Secondly, a forward engineering has to be done to restructure the source code of the application according to the new architecture. The new source code was migrated to COBOL9x. Because of using PROGRES and TXL an adaptation to other reengineering problems is possible.

Another project which is investigated within our group is the E-CARES project where PROGRES is widely used deals with real-time systems in telecommunications. It is targeted at developing methods, concepts, and tools to understand and restructure complex systems such as Ericsson's Mobile Service Switching Centers. PROGRES is used for modelling the software architecture and the system architecture. This project has started at the end of the year and first successes have already been achieved due to the use of PROGRES.

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 [Plu99b], 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 first part of the habilitation thesis [Plu99a] also surveys term graph rewriting, while the second part studies termination and confluence—the key properties for term and term graph rewriting—in the setting of general graph transformation.

The contents of the paper [AKP00] was described in the First Progress

Report.

Term graphs play a central role in the functorial presentation of multi-algebras, recently proposed by Corradini and Gadducci. Multi-algebras allow to model nondeterminism in an algebraic framework by interpreting operators as functions from individual arguments to sets of possible results. In [CG99a] and its full version [CG00], the authors propose a functorial presentation of various categories of multi-algebras and partial algebras, analogous to the classical presentation of algebras over a signature Σ as cartesian functors from the algebraic theory over Σ to **Set**. They introduce two different notions of theory over a signature, both having a structure weaker than cartesian, and they consider functors from them to **Rel** or **Pfn**, the categories of sets and relations or partial functions, respectively. In particular, the theory used for classifying multialgebras is the *gs-monoidal* theory of a signature, which was shown in previous works by the same authors to represent faithfully term graphs over the signature. Based on this observation, it is argued that the natural generalization of usual terms for multi-algebras are term graph. This idea is developed further in [CGK00], where a calculus for term graph equational deduction is presented and its soundness is proved. The authors argue that term graph equations are more expressive than standard equations for the specification of multi-algebras, but they admit that inequations (interpreted as inclusions of the corresponding relations) are more adequate for the specification of non-determinism.

Paper [CG99b] has been accepted for publication in *Theoretical Informatics and Applications*; this is a full version of paper [CG98] which was already described in the Second Annual Report.

4.3.7 Application to Neural Networks

In [BFK00], attributed graph transformation is used to model neural networks. The paper follows a series of previous papers modeling neural networks and showing their termination. This paper emphasizes the labeling of the network structure which is the key in modeling a neural network. A common problem in many graph transformation approaches is the relabeling of objects. An existing solution is for example the introduction of auxiliary graph structures. In [BFK00], a double pushout approach to attributed graph transformation is presented that allows an intuitive way of relabeling due to a partial attribution.

Different learning algorithms based on learning from examples based on a set of graph rewrite rules are described in [FKB00]. Starting from either a very general or a very special rule set which is modeled as graph, two to three basic rewrite rules are applied until a rule graph explaining all examples is reached. The rewrite rules can also be used to model the corresponding hypothesis space as they describe partial relations between different rule set graphs. The possible paths, algorithms can take through the hypothesis space can be described as application sequences. This schema is applied to general learning algorithms as well as to fuzzy rule learning algorithms.

To generate Fuzzy Rules from given data several algorithms have been proposed in the last decade. These algorithms can be described using transformation rules on Fuzzy Petri nets which is done in [FKB00]. The different resulting Fuzzy Petri nets can be compared and a hierarchy of possible Fuzzy Petri nets is proposed. Easy transformation rules describe the differences between the nets. Possible optimal Fuzzy Petri nets for a given data set are proposed and proofs whether a given net is optimal along these lines are sketched.

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.

[AKP00] Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Bisimilarity in term graph rewriting. *Information and Computation*, 156(1/2):2–24, 2000.

[AP99] Peter Achten and Rinus Plasmeijer. Some implementation issues on the Clean Object I/O library — Draft version. In P. Koopman and C. Clack, editors, *Proceedings of the 11th International Workshop on the Implementation of Functional Languages, IFL'99*, pages 130–143. Lochem, The Netherlands, September 7-10 1999. Final paper will appear in LNCS.

[AW00] Peter Achten and Martin Wierich. A Tutorial to the Clean Object I/O Library — Version 1.2. Technical Report CSI-R0003, Computing Science Institute, Faculty of Mathematics and Informatics, University of Nijmegen, February 2000.

[Bal00] P. Baldan. *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars.* Doctoral dissertation, Dipartimento di Informatica, Pisa, 2000.

[Bar99] R. Bardohl. *GenGEd - Visual Definition of Visual Languages based on Graph Transformation.* Doctoral dissertation, Fachbereich Informatik, Technische Universität Berlin, 1999.

[BCE+99] P. Baldan, A. Corradini, H. Ehrig, M. Loewe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformations. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 3: Concurrency, Parallelism, and Distribution*, pages 109 – 187. World Scientific, 1999.

[BCM00] P. Baldan, A. Corradini, and U. Montanari. Unfolding of double-pushout graph grammars is a coreflection. In H. Ehrig, G. Engels, H.-J. Kreowsky, and G. Rozenberg, editors, *Proceedings TAGT'98*, volume 1764 of *LNCS*, pages 145–163. Springer Verlag, 2000.

[BE00] R. Bardohl and H. Ehrig. Conceptual Model of the Graphical Editor GenGEd. In Ehrig et al. [EEKR00].

[BEMW00] G. Busatto, G. Engels, K. Mehner, and A. Wagner. A framework for adding packages to graph transformation approaches. In Ehrig et al. [EEKR00], pages 352–367.

[BER00] R. Bardohl, C. Ermel, and L. Ribeiro. Towards Visual Specification and Animation of Petri Net Based Models. In Ehrig and Taentzer [ET00], pages 22–31.

[BFK00] M. Berthold, I. Fischer, and M. Koch. Attributed graph transformation with partial attribution and its application to neural networks. In Ehrig and Taentzer [ET00], pages 171–179.

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

[BM99a] R. Bruni and U. Montanari. Cartesian closed double categories, their lambda-notation, and the pi-calculus. In *LICS'99, 14th Annual IEEE Symposium on Logic in Computer Science*, pages 246–265. IEEE Computer Society, 1999.

[BM99b] R. Bruni and U. Montanari. Zero-safe nets: Composing nets via transition synchronization. In H. Weber, H. Ehrig, and

W. Reisig, editors, *Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, pages 43–80. Fraunhofer Gesellschaft ISST, 1999.

[BM00a] R. Bruni and U. Montanari. Executing transactions in zero-safe nets. In *ICATPN2000, Int. Conf. on Application and Theory of Petri Nets*, LNCS. Springer Verlag, 2000. To appear.

[BM00b] R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Information and Computation*, 156:46–89, 2000.

[BMS00] R. Bruni, U. Montanari, and V. Sassone. Open ended systems, dynamic bisimulation and tile logic. In *IFIP TCS2000, Exploring New Frontiers of Theoretical Informatics*, LNCS. Springer Verlag, 2000. 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 II: Applications, Languages and Tools, pages 105–180. World Scientific, 1999.

[BNS00] R. Bardohl, M. Niemann, and M. Schwarze. GenGEd: A Development Environment for Visual Languages. In Nagl et al. [NSM00].

[BP00] P. Bottoni and F. Parisi Presicce. Patterns on graphs. In Ehrig and Taentzer [ET00], pages 180–188.

[BPS99] P. Bottoni, F. Parisi-Presicce, and M. Simeoni. On a uniform representation of transformation systems. In *Proc. Applications of Graph Transformations with Industrial Relevance (ACTIVE '99)*, 1999.

[BPS00] P. Bottoni, F. Parisi-Presicce, and M. Simeoni. From formulae to rewriting systems. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number 1764 in LNCS, pages 267–280. Springer, 2000.

[BTMS99] R. Bardohl, G. Taentzer, M. Minas, and A. Schürr. Application of Graph Transformation to Visual Languages. In Ehrig et al. [EEKR99], pages 105–180.

[CG98] A. Corradini and F. Gadducci. Rewriting on cyclic structures. Technical Report TR-98-05, Dipartimento di Informatica, Pisa, 1998.

[CG99a] A. Corradini and F. Gadducci. Functorial semantics for multi-algebras. In J. L. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques*, volume 1589 of *LNCS*, pages 78–90. Springer Verlag, 1999.

[CG99b] A. Corradini and F. Gadducci. Rewriting on cyclic structures: Equivalence between the operational and the categorical description. *Theoretical Informatics and Applications*, 33:467–493, 1999.

[CG00] A. Corradini and F. Gadducci. A functorial semantics for multi-algebras and partial algebras, with applications to syntax. *Theoretical Computer Science*, 2000. To appear.

[CGK00] A. Corradini, F. Gadducci, and W. Kahl. Term graph syntax for multi-algebras. Technical Report TR-00-04, Dipartimento di Informatica, 2000.

[CMP99] Sitt Sen Chok, Kim Marriott, and Tom Paton. Constraint-based diagram beautification. In *Proc. 1999 IEEE Symp. on Visual Languages, Tokyo, Japan*, 1999.

[CNG99] K. Cremer, M. Nagl, and S. Gruner. Graph transformation based integration tools: Application to chemical process engineering. In Ehrig et al. [EEKR99].

[Cre99] Katja Cremer. *Anwendung von Graphentechnik zum Reverse Engineering und Reengineering.* Doctoral dissertation, RWTH Aachen, 1999.

[Cre00] Katja Cremer. Graph-based reverse engineering and reengineering tools. In Manfred Nagl, Andy Schürr, and Manfred Münch, editors, *Application of Graph Transformations with Industrial Relevance*, LNCS 1779, pages 95–110. Springer, 2000.

[DHK00a] R. Depke, R. Heckel, and J.M. Küster. Integrating visual modeling of agent-based and object-oriented systems. In *Proc. Agents2000, Barcelona*, 2000. To appear.

[DHK00b] R. Depke, R. Heckel, and J.M. Küster. Modeling agent-based systems with graph transformation and UML: From requirement specification to object-oriented design. In *GRATRA 2000: Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems*, pages 87–95, Berlin, Germany, March 2000.

[DHP00a] Frank Drewes, Berthold Hoffmann, and Detlef Plump. Hierarchical graph transformation. In *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2000)*, volume 1784 of *Lecture Notes in Computer Science*, pages 98–113. Springer-Verlag, 2000.

[DHP00b] Frank Drewes, Berthold Hoffmann, and Detlef Plump. Hierarchical graph transformation. Informatik-Bericht 1/00, Universität Bremen, 2000. 33 pages.

[DK00] Frank Drewes and Peter Knirsch. Treebag – a short presentation. In M. Nagl and A. Schürr, editors, *Proc. Applications of Graph Transformation with Industrial Relevance (AGTIVE'99)*, Lecture Notes in Computer Science, 2000. To appear.

[DKK00] Frank Drewes, Renate Klempien-Hinrichs, and Hans-Jörg Kreowski. Table-driven and context-sensitive collage languages. In G. Rozenberg and W. Thomas, editors, *Proc. Developments in Language Theory (DLT'99)*. World Scientific, 2000. To appear.

[DKKK00] Frank Drewes, Peter Knirsch, Hans-Jörg Kreowski, and Sabine Kuske. Graph transformation modules and their composition. In *Proc. Workshop on Applications of Graph Transformations With Industrial Relevance*, Lecture Notes in Computer Science, 2000. To appear.

[dMvE99] Maarten de Mol and Marko van Eekelen. A Prototype Dedicated Theorem Prover for Clean. Technical Report CSI-R9913, Computing Science Institute, University of Nijmegen, 1999.

[dMvE00] Maarten de Mol and Marko van Eekelen. A Proof Tool Dedicated to Clean. In Nagl et al. [NSM00].

[Dre98a] Frank Drewes. TREEBAG—a tree-based generator for objects of various types. Report 1/98, Univ. Bremen, 1998.

[Dre98b] 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.

[Dre99] Frank Drewes. Tree-based generation of languages of fractals. Report 2/99, Univ. Bremen, 1999.

[Dre00a] Frank Drewes. Some remarks on the generative power of collage grammars and chain-code grammars. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Sixth Intl. Workshop on Theory and Application of Graph Transformations (TAGT'98)*, volume 1764 of *Lecture Notes in Computer Science*, pages 1–14, 2000.

[Dre00b] Frank Drewes. Tree-based picture generation. *Theoretical Computer Science*, 2000. To appear.

[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.

[EEKR00] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Theory and Application of Graph Transformations (TAGT '98), Selected Papers*, volume 1764 of *Lecture Notes in Computer Science.* Springer-Verlag, 2000.

[EGPP99] H. Ehrig, M. Gajewsky, and F. Parisi Presicce. High-Level Replacement Systems with Applications to Algebraic Specifications and Petri Nets. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution*, chapter 4. World Scientific, 1999.

[EHL+00] H. Ehrig, R. Heckel, M. Llabres, F. Orejas, J. Padberg, and G. Rozenberg. Double-pullback graph transitions: A rule-based framework with incomplete information. In Ehrig et al. [EEKR00].

[EKMR99] Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution.* World Scientific, 1999.

[ET00] Hartmut Ehrig and Gabi Taentzer, editors. *Proc. Joint APPLIGRAPGH/GETGRATS Workshop on Graph Transformation Systems(GRATRA 2000).* Technical report 2000-2, Fachbereich Informatik, Technische Universität Berlin, 2000.

[FKB00] Ingrid Fischer, Manuel Koch, and Michael R. Berthold. On Generating Fuzzy Models with Fuzzy Petri Nets. In Nagl et al. [NSM00].

[FKT00] Ingrid Fischer, Manuel Koch, and Gabriele Taentzer. Local Views on Distributed Systems and their Communication. In Ehrig et al. [EEKR00], pages 164–178.

[GEMT00] M. Goedicke, B. Enders, T. Mayer, and G. Taentzer. Tool support for viewpoint-oriented software development: Towards integration of multiple perspectives by distributed graph transformation. In Nagl et al. [NSM00].

[GHK00] F. Gadducci, R. Heckel, and M. Koch. A fully abstract model for graph-interpreted temporal logic. In Ehrig et al. [EEKR00].

[GHL99] F. Gadducci, R. Heckel, and L. Llabrés. A bi-categorical axiomatisation of concurrent graph rewriting. In M. Hofmann, D. Pavlović, and P. Rosolini, editors, *Proc. 8th Conference on Category Theory and Computer Science (CTCS'99), Edinburgh, UK*, volume 24 of *ENTCS*. Elsevier Science, September 1999. `http://www.elsevier.nl/locate/entcs`.

[GM00] 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, 2000. To appear.

[GMET00] M. Goedicke, T. Meyer, B. Enders, and G. Taentzer. Viewpoint-oriented software development: Support for integrating multiple perspectives by distributed graph transformation. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 1785 in LNCS. Springer, 2000.

[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. 4th IEEE International Symposium on Requirements Engineering (RE'99), June 7-11, 1999, University of Limerick, Ireland.* IEEE Computer Society, 1999. ISBN 0-7695-0188-5.

[GMT00] M. Goedicke, T. Meyer, and G. Taentzer. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. In Ehrig et al. [EEKR00].

[GPS00] M. Große–Rhode, F. Parisi Presicce, and M. Simeoni. Refinements of graph transformation systems via rule expressions. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. 6th International Workshop on Theory and Applications of Graph Transformation (TAGT'98)*, number 1764 in LNCS, pages 368 – 382. Springer, 2000.

[GPST99] M. Große–Rhode, F. Parisi Presicce, M. Simeoni, and G. Taentzer. Modeling distributed systems by modular graph transformation based on refinement via rule expressions. in Proc. of "Applications of Graph Transformations with Industrial Relevance", Kerkrade, The Netherlands, September, 1999.

[GR00] M. Grosse-Rhode. Using a formal reference model for consistency checking and integration of uml diagrams. In *Proc. 5th World Conference on Design and Process Technology, Dallas, Texas, USA, 4-8 July*, 2000. To appear.

[Gri96] Peer Griebel. *Parcon - Paralleles Lösen von grafischen Constraints.* Dissertation, Universität Paderborn, February 1996.

[Gru99] Stefan Gruner. *Eine schematische und grammatische Korrespondenzmethode zur Spezifikation konsistent verteilter Datenmodelle.* Doctoral dissertation, RWTH Aachen, 1999.

[Gru00a] S. Gruner. A combined graph schema and graph grammar approach to consistency in distributed modeling. In Nagl et al. [NSM00].

[Gru00b] S. Gruner. Meta typing is compatible to the typed SPO approach. In Ehrig and Taentzer [ET00], pages 50–58.

[GSP99] M. Große–Rhode, M. Simeoni, and F. Parisi Presicce. Refinements and modules for typed graph transformation systems. In J.L.Fiadeiro, editor, *Proc. WADT'98 (Workshop on Algebraic Development Techniques), at ETAPS'98, Lisbon, April*, number 1589 in LNCS, pages 138 – 151. Springer, 1999.

[HAKP99a] Zoltan Horvath, Peter Achten, Tamas Kozsik, and Rinus Plasmeijer. Proving the Temporal Properties of the Unique World. In *Software Technology, Fenno-Ugric Symposium FUSST'99*, CS 104/99, pages 113–125, Tallin, 1999.

[HAKP99b] Zoltan Horvath, Peter Achten, Tamas Kozsik, and Rinus Plasmeijer. Verification of the Temporal Properties of Dynamic Clean Processes. In P. Koopman and C. Clack, editors, *Proceedings of the 11th International workshop on the Implementation of Functional Languages, IFL'99*, pages 203–218, Lochem, The Netherlands, September 7-10 1999.

[HEET99a] Reiko Heckel, Hartmut Ehrig, Gregor Engels, and Gabriele Taentzer. Classification and comparison of module concepts for graph transformation systems. In H. Ehrig, G. Engels, J.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, pages 669 – 690. World Scientific, 1999.

[HEET99b] Reiko Heckel, Hartmut Ehrig, Gregor Engels, and Gabriele Taentzer. A view-based approach to system modeling based on open graph transformation systems. In Ehrig et al. [EEKR99], pages 639–668.

[HHKK00] Reiko Heckel, Berthold Hoffmann, Peter Knirsch, and Sabine Kuske. Simple modules for GRACE. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Proc. Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 383–395, 2000.

[HLEO00] R. Heckel, L. Llabrés, H. Ehrig, and F. Orejas. Concurrency of double-pullback transitions. In Ehrig and Taentzer [ET00].

[HM99] D. Hirsch and U. Montanari. Consistent transformations for software architecture styles of distributed systems. In G. Stefanescu, editor, *Workshop on Distributed*

Systems, volume 28 of *ENTCS*, 1999. Available from http://www.elsevier.nl/locate/entcs/volume28.html.

[HM00] D. Hirsch and U. Montanari. Higher-order hyperedge replacement systems and their transformations. In *Proceedings GRATRA 2000*, 2000. Technical Report n. 2000-2, Technical University Berlin.

[HMP99] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. Technical Report 7/99, Fachbereich Informatik, Universität Oldenburg, 1999. 68 pages.

[HMP00a] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout approach with injective matching. In Ehrig et al. [EEKR00], pages 103–116.

[HMP00b] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 2000. To appear.

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

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

[Jan99] D. Janssens. Actor grammars and local actions. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3, pages 57–106. World Scientific, 1999.

[JK00] Hélène Jacquet and Renate Klempien-Hinrichs. Node replacement in hypergraphs: Translating NCE rewriting into the pullback approach. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Sixth Intl. Workshop on Theory and Application of Graph Transformations (TAGT'98)*, volume 1764 of *Lecture Notes in Computer Science*, pages 117–130. Springer, 2000.

[JV99] D. Janssens and N. Verlinden. Algebraic properties of LAS processes. Technical Report 99-13, University of Antwerp - UIA, Dept. of Math. and Comp. Sci., 1999.

[JV00] D. Janssens and N. Verlinden. A framework for ESM and NLC: Local action systems. In *6th International Workshop on Theory and Aplication of Graph Transformation*, volume 1764, pages 194–214. Springer Verlag, 2000.

[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 *Proceedings of the Ninth International Workshop on Software Specification and Design*, pages 77–86. IEEE Computer Society, 1998.

[KK99] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units and modules. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. II: Applications, Languages and Tools*, chapter 15, pages 607–638. World Scientific, 1999.

[KK00a] Peter Knirsch and Hans-Jörg Kreowski. A note on modeling agent systems by graph transformation. In M. Nagl and A. Schürr, editors, *Proc. Applications of Graph Transformation with Industrial Relevance (AGTIVE'99)*, Lecture Notes in Computer Science. Springer, 2000. To appear.

[KK00b] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units with interleaving semantics. *Formal Aspects of Computing*, 2000. To appear.

[KKT99] Renate Klempien-Hinrichs, Hans-Jörg Kreowski, and Stefan Taubenberger. Correct translation of mutually recursive function systems into TOL collage grammars. In G. Ciobanu and Gh. Păun, editors, *Proc. 12th Intl. Symposium on Fundamentals of Computation Theory (FCT'99)*, volume 1684 of *Lecture Notes in Computer Science*. Springer, 1999.

[Kle00a] Renate Klempien-Hinrichs. The generative power of context-free node rewriting in hypergraphs. *Grammars*, 2000. To appear.

[Kle00b] Renate Klempien-Hinrichs. Normal forms for context-free node-rewriting hypergraph grammars. In Ehrig and Taentzer [ET00].

[KM00] Oliver Köth and Mark Minas. Generating diagram editors providing free-hand editing as well as syntax-directed editing. In *Proc. International Workshop on Graph Transformation (*GRA-TRA *2000), Berlin*, March 2000.

[KNS99] P. Klein, M. Nagl, and A. Schürr. IPSEN tools. In Ehrig et al. [EEKR99], pages 215–265.

[Koc99] Manuel Koch. *Integration of Graph Transformation and Temporal Logic for the Specification of Distributed Systems.* Doctoral dissertation, TU Berlin, 1999.

[Koc00a] M. Koch. Compositional specification of distributed systems by distributed graph transformation. In G. Hommel, editor, *Proc. Int. Workshop on Communication-Based Systems.* Kluwer, 2000.

[Koc00b] M. Koch. Single pushout approach to distributed graph transformation. In Ehrig and Taentzer [ET00], pages 249–256.

[Kus00a] Sabine Kuske. More about control conditions for transformation units. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 323–337, 2000.

[Kus00b] Sabine Kuske. *Transformation Units — A Structuring Principle for Graph Transformation Systems.* Doctoral dissertation, Universität Bremen, Fachbereich Mathematik und Informatik, 2000. 159 pages.

[MCF98] Kim Marriott, Sitt Sen Chok, and Alan Finlay. A tableau based constraint solving toolkit for interactive graphical application. In *4th International Conference on Principles and Practice of Constraint Programming (CP'98), Pisa, Italy*, pages 340–354, October 1998.

[MS99] Manfred Münch and Andy Schürr. Leaving the visual languages ghetto. In *IEEE Symposium on Visual Languages*, pages 148–155. IEEE Computer Society, 1999.

[Mün00] Manfred Münch. Programmed graph rewriting system PROGRES. In Manfred Nagl, Andy Schürr, and Manfred

Münch, editors, *Application of Graph Transformations with Industrial Relevance*, volume 1779 of *Lecture Notes in Computer Science*, pages 441–448. Springer-Verlag, 2000.

[NSM00] Manfred Nagl, Andy Schürr, and Manfred Münch, editors. *Proc. Applications of Graph Transformations with Industrial Relevance (AGTIVE'99)*, volume 1779 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[NZ00] Jörg Niere and Albert Zündorf. Using fujaba for the development of production control systems. In Manfred Nagl, Andy Schürr, and Manfred Münch, editors, *Application of Graph Transformations with Industrial Relevance*, volume 1779 of *Lecture Notes in Computer Science*, pages 181–192. Springer-Verlag, 2000.

[Pil99] M.R.C. Pil. Dynamic Types and Type Dependent Functions. In K. Hammond, A.J.T. Davie, and C. Clack, editors, *Implementation of Functional Languages (IFL '98)*, LNCS 1595, pages 169–185, London, UK, 1999. Springer-Verlag.

[Plu99a] Detlef Plump. *Computing by Graph Rewriting.* Habilitation thesis, Universität Bremen, Fachbereich Mathematik und Informatik, 1999. 193 pages.

[Plu99b] Detlef Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, chapter 1. World Scientific, 1999.

[PW00] F. Parisi-Presicce and A. Wolf. Foundation for software configuration management policies using graph transformation. In T.Maibaum, editor, *Foundations of Software Engineering (FASE 2000)*, number 1783 in LNCS, pages 304–318. Springer, 2000.

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

[Rad00] Ansgar Radermacher. Dito - a distribution tool based on graph rewriting. In Manfred Nagl, Andy Schürr, and Manfred Münch, editors, *Application of Graph Transformations with Industrial*

Relevance, LNCS 1779, pages 465–472. Springer Verlag, Berlin, 2000.

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

[Sim00] M. Simeoni. *A Categorical Approach to Modularization of Graph Transformation Systems using Refinement.* Doctoral dissertation, University of Rome, Italy, 2000.

[SWZ97] Andy Schürr, Andreas J. Winter, and Albert Zündorf. Progres: Language and environment. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools*, pages 487–550. World Scientific, Singapore, 1997.

[Tae99] G. Taentzer. Adding visual rules to object-oriented modeling techniques. In R. Mitchell, A. C. Wills, J. Bosch, and B. Meyer, editors, *Technology of Object-Oriented Languages and Systems (TOOLS 29), June 7 - 10, 1999, Nacy, France.* IEEE Computer Society, 1999.

[TE00] A. Tsiolakis and H. Ehrig. Consistency Analysis of UML Class and Sequence Diagrams using Attributed Graph Grammars. In Ehrig and Taentzer [ET00], pages 77–86.

[TER99] G. Taentzer, C. Ermel, and M. Rudolf. The AGG-approach: Language and tool environment. In Ehrig et al. [EEKR99], pages 551–604.

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

[TGM99] G. Taentzer, M Goedicke, and T. Meyer. Dynamic accomodation of change: Automated architecture configuration of distributed systems. In *Automated Software Engineering'99, Cocoa Beach, Florida, USA*. IEEE Computer Society, 1999.

[vG99] John van Groningen. Optimizing Recursive Functions Yielding Multiple Results in Tuples in a Lazy Functional Language. In P. Koopman and C. Clack, editors, *Proceedings of the 11th International workshop on the Implementation of Functional Languages, IFL'99*, Lochem, The Netherlands, September 7-10 1999. To appear as selected paper in LNCS.

[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 on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools*, volume 2, pages 321–368. World Scientific, Singapore, 1999.

[Win99] Andreas Winter. *Visuelles Programmieren mit Graphtransformationen.* Doctoral dissertation, RWTH Aachen, Aachener Beiträge zur Informatik, Band 27, 1999.

[WJS00] Bernhard Westfechtel, Dirk Jäger, and Ansgar Schleicher. Ahead: A graph-based system for modeling and managing development processes. In *Application of Graph Transformations with Industrial Relevance*, volume 1779 of *LNCS*, pages 325–340. Springer-Verlag, 2000.

[WPA99] Mike Wiering, Rinus Plasmeijer, and Peter Achten. Using Clean for Platform Games - Draft. In P. Koopman and C. Clack, editors, *11th International Workshop on the Implementation of Functional Languages, IFL'99*, pages 144–155, Lochem, The Netherlands, September 7-10 1999. Final paper will appear in LNCS.