Appligraph
Applications of Graph Transformation

ESPRIT Working Group No. 22565

Final Report

Hans-Jörg Kreowski and Detlef Plump
(editors)

July 2002

Contents

1 Introduction

This report summarizes the activities of the ESPRIT Working Group APPLIGRAPH, covering the period from April 1, 1997, to April 30, 2002. The principal objective of this Working Group was 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 was organized according to the following three research areas:

- *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 were investigated, like typing, modularity, refinement, parallelism, concurrency, distribution, optimisation, and correctness.

The area of *Tool Issues* was 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 was 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 Teams and Scientific Correspondents

The APPLIGRAPH Working Group was coordinated by Hans-Jörg Kreowski (University of Bremen, Germany) and managed by Detlef Plump (University of York, United Kingdom). Research teams at the following sites participated in APPLIGRAPH:

Aachen University of Technology (Germany)
Team leader: Manfred Nagl

University of Antwerp (Belgium)
Team leader: Dirk Janssens

Technical University of Berlin (Germany)
Team leaders: Hartmut Ehrig and Herbert Weber

University of Bremen (Germany)
Team leaders: Otthein Herzog and Hans-Jörg Kreowski

University of Leiden (The Netherlands)
Team leader: Grzegorz Rozenberg

University of Nijmegen (The Netherlands)
Team leader: Rinus Plasmeijer

University of Paderborn (Germany)
Team leader: Gregor Engels

University of Pisa (Italy)
Team leader: Ugo Montanari

University of Rome (Italy)
Team leader: Francesco Parisi-Presicce

APPLIGRAPH cooperated with the following scientific correspondents: 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), Tero Harju (Turku, Finland), Neil Jones (Copenhagen, Denmark), Simon Peyton Jones (Glasgow, Scotland), Michael Löwe (Hannover, Germany), Alfonso Pierantonio (L'Aquila, Italy), Wilhelm Schäfer (Paderborn, Germany), Hans-Jürgen Schneider (Erlangen, Germany), Andy Schürr (München, Germany), Gabriel Valiente Feruglio (Barcelona, Spain), Albert Zündorf (Paderborn, Germany)

3 Summary of Activities

3.1 Research

The results of the research done in APPLIGRAPH are documented in 45 papers published or accepted for publication in journals and books, 5 books, 210 papers published or accepted for publication in conference and workshop proceedings, 17 doctoral dissertations and 28 technical reports. (See also Subsection 3.3 for the involvement of APPLIGRAPH in the edition of books and special journal issues.) The scientific results are described in Section 4 and references to publications are given at the end of this report.

3.2 Meetings

3.2.1 Workshops and Related Events

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

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

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

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.

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

UniGra 2001 – Workshop on Uniform Approaches to Graphical Process Specification Techniques. This workshop was co-organized by APPLIGRAPH as well. It was held in Genova (Italy) on March 31 and April 1, 2001, as a satellite event of the ETAPS 2001 joint conference.

GT-VMT 2001 – Second International Workshop on Graph Transformation and Visual Modeling Techniques. This workshop co-organized by APPLIGRAPH took place on Crete (Greece) on July 12-13, 2001, as a satellite event of the ICALP 2001 conference.

International Colloquium on Graph Transformation and DNA Computing. This colloquium took place on February 14, 2002, at the Technical University of Berlin to celebrate the award of an honorary doctorate to Prof. Grzegorz Rozenberg (Leiden University).

AGT 2002 – Final APPLIGRAPH Workshop on Applied Graph Transformation. This final event of the Working Group took place as an open satellite workshop of the ETAPS 2002 joint conference in Grenoble (France) from April 6 to April 14, 2002.

3.2.2 Subgroup Meetings

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

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

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

Kerkrade 1999. 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.

Bremen 1999. 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.

Paderborn 2000. This meeting took place September 5–6, 2000, at the University of Paderborn (Germany). Its purpose was to discuss common XML-based exchange formats for graphs and graph transformation systems.

Bremen 2001. This meeting was held March 1-2, 2001, at the University of Bremen (Germany). It continued the discussion and development on XML-based exchange formats for graphs and graph transformation systems.

Erlangen 2002. This meeting on XML-based graph exchange formats took place on March 7-8, 2002, at the University of Erlangen (Germany).

3.2.3 Team Leader and Business Meetings

The purpose of the following meetings was to discuss various administrative and financial issues of the Working Group, and to plan future events.

First Team Leader Meeting. This meeting took place on June 26, 1997, during the first APPLIGRAPH Workshop.

Second Team Leader Meeting. This meeting took place on March 3, 1998, during the European School on Graph Transformation.

Business Meeting. An APPLIGRAPH Business Meeting took place on November 17, 1998, during TAGT '98.

Third Team Leader Meeting. This meeting took place on November 19, 1998, during TAGT '98.

Fourth and Fifth Team Leader Meeting. These meetings took place during the AGTIVE Symposium and the GRATRA 2000 Workshop, respectively.

Sixth Team Leader Meeting. This meetings was held at ETAPS 2001 in Genova (Italy), as part of the meeting of the Steering Committee of the International Conference on Graph Transformation (ICGT).

3.3 Information Dissemination

One of the main goals of the APPLIGRAPH Working Group was 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 goal was approached by the activities described below.

3.3.1 Handbook of Graph Transformation

Three volumes of the *Handbook of Graph Grammars and Computing by Graph Transformation* appeared with World Scientific Publishers. The first volume [Roz97] is on *Foundations*, the second volume [EEKR99] is on *Applications, Languages, and Tools* and the third volume [EKMR99] is on *Concurrency, Parallelism, and Distribution.* The three volumes together cover the research areas of APPLIGRAPH.

3.3.2 Workshop Proceedings

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 [ET00b] and `http://tfs.cs.tu-berlin.de/gratra2000`.

Proceedings of GT-VMT 2000. Extended abstracts of the papers presented at the Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2000) have been published by Carleton Scientific in a volume collecting the contributions to all satellite workshops of ICALP 2000, see [CH00].

Proceedings of UniGra 2001. The proceedings of the Workshop on Uniform Approaches to Graphical Process Specification Techniques (Uni-

Gra 2001) appeared as Volume 44, Issue 4 of *Electronic Notes in Theoretical Computer Science* (Elsevier Science, 2001), see [EEP01].

Proceedings of GT-VMT 2001. The proceedings of the Second International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2001) appeared as Volume 50, Issue 3 of *Electronic Notes in Theoretical Computer Science* (Elsevier Science, 2002), see [BPT01].

Proceedings of AGT 2002. The proceedings of the Workshop on Applied Graph Transformation (AGT 2002) were published by the University of Grenoble in a series of workshop proceedings devoted to the satellite events of the ETAPS 2002 joint conference, see [KK02].

3.3.3 Special Journal Issues

Selected papers of the Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000) appeared as special journal issues *Mathematical Structures in Computer Science* 12:4 (Cambridge University Press, 2002) and *Science of Computer Programming* 44:2 (Elsevier Science, 2002).

3.3.4 Miscellaneous

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

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).

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`.

3.4 Further Activities

To establish a sustained promotion of the area of graph transformation, APPLIGRAPH supported the Steering Committee of the International Conference on Graph Transformation (ICGT). Its 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.

3.5 Special Highlight

In February 2002 Grzegorz Rozenberg, one of the APPLIGRAPH team leaders, has been awarded an honorary doctorate by the Technical University of Berlin for his scientific achievements including his work on graph transformation.

4 Scientific Results

The following description of APPLIGRAPH'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

Transformation units and transformation modules The notions of transformation unit and transformation module are a means for constructing large graph transformation systems from small ones in a structured and systematic way. A transformation unit comprises a set of rules, descriptions of initial and terminal graphs, a control condition, and a set of imported transformation units. Semantically, it transforms initial graphs to terminal ones by interleaving ordinary direct derivation steps with calls of imported transformation units such that the control condition is obeyed. The concept of transformation units is independent of a particular graph transformation approach and, therefore, it may enhance the usefulness of graph transformations in many contexts.

The notion of a transformation unit together with its interleaving semantics is presented and thoroughly studied in [KK99b] and in the doctoral thesis [Kus00b]. After illustrating the concept of transformation units with an example from the area of VLSI theory, [KK99b] presents 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.

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

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

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

In [Kus01] the structuring concept of transformation units is employed for the formalization of the dynamic semantics of UML state machines. [Kus02] extends transformation units by formal parameters. The resulting parameterized transformation units are used in [KKK02] for a first case study which models the logistic Pickup-and-Delivery Problem based on graph transformation.

In [KK99a] 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 compound graph transformations that call rules and other transformation units which are defined or imported by the surrounding module. Moreover, they allow pre- and post-conditions to be specified. Semantically, each module specifies a binary relation on graphs for each exported item. The transformation units of modules are a simplified version of those in [KK99a], because they are relieved from their interfaces and local rules.

In [KK00b] transformation units and transformation modules are generalized in such a way that they can be used for the structuring of arbitrary rule-based systems. Moreover, the interleaving semantics is relaxed in the sense that the imported units may now operate on local components of actual configurations rather than on the global objects. The paper [KK00c] proposes an approach to design modular systems with heterogeneous components, i.e. the transformation units and the transformation modules of a system may be based on different semantic domains and different types of rules and rule applications as long as the formats of the configurations are compatible with each other.

Transformation units are the basic components of the graph- and rule-centered specification language GRACE, which is being designed to support the modeling of a wide spectrum of graph and graphical processes in a structured and uniform way. In [KBK01], some basic ideas and features of GRACE are sketched, and [KBK$^+$02] illustrates the structuring concepts of GRACE, including transformation units, formal parameter units, transformation modules, and their composition to modular systems.

In [Kre02], the basic features of graph transformation are introduced as a kind of sight-seeing tour of the computational landscape which is based on the application of rules to graphs. In particular, constructing a functional relation on graphs by graph transformation and a novel type of parallel graph transformation are studied.

Hierarchical graph transformation If systems are specified by graph transformation, large graphs should be structured in order to be comprehensible. The papers [DHP02, 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.

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. This approach towards the definition of a general-purpose hierarchical graph data model has been continued and formalized [BKK01], where a hierarchical graph model is introduced which allows to structure large graphs into small components. The model is equipped with a rule-based transformation concept in order to describe dynamic system behavior.

In [BH01], a generic model of hierarchical graphs and their transformation is presented. This model is used to compare the power of other existing approaches to hierarchical graphs. In [Bus02b], a model of hypermedia based on hierarchical graph transformation is presented. In [Bus02a], a generic model of hierarchical graphs and their transformation is introduced. This model is used for defining concrete hierarchical graph models, for investigating related approaches, and in a practical application.

In [Hof01], extensions of graphs and graph transformation that are important for programming with graphs are studied: On the one hand, graphs are extended by a notion of hierarchy that supports value composition, and hierarchical graph transformation is defined in an intuitive way that resembles term rewriting. On the other hand, admissible shapes for hierarchical graphs are required to be specified by context-free graph grammars, in order to set up a type discipline for shapely hierarchical graph transformation.

In [HM01c], a new computational model for rule-based programming with graphs and diagrams is described. Using existing nesting concepts for graphs, this model defines an intuitive way of nested graph transformation that is based on variable matching. Shape rules are introduced for specifying structural consistency conditions on nested graphs. Shape rules set up a decidable type discipline for a refined model of shapely nested graph transformation. Since the refined model is compatible with the diagram editor DiaGen, it can be extended by costomizable diagram interfaces so that it specifies rule-based diagram transformation.

In [DHM02], it is shown that the construction of shapely nested graph transformation steps can be reduced to solving the subgraph isomorphism and variable matching problem for the components of a structured graph, and reestrictions of the transformation rules are devised that improve efficiency.

Local action systems Modularity aspects for graph rewriting based on node replacement and embedding have been investigated in the framework of Local Action Systems (LAS). These were introduced in [Jan99], and it was shown in [JV00] that several other types of graph rewriting, including NLC and NCE, can be considered as special cases of them. In the LAS approach, productions are viewed as composed of local actions. When applied to actor systems, e.g., this corresponds to the fact that an event (actor grammar production) is composed of more basic instructions, combined into a script. Thus productions as such embody already a rather primitive form of modularity. On a slightly higher level, each LAS process, composed of productions, corresponds again to a production. Hence a step, or production occurrence, of a high-level system description can be refined into a process consisting of a combination of lower level steps. In cooperation with M. Simeoni (Rome), it was shown that this idea is captured by so-called Cat-modules: in this framework a module consists of three systems (export, import and body) that are connected by suitable refinement morphisms. It was shown that the general notion of Cat-modules can be instantiated to yield a module concept for Local Action Systems: suitable refinement morphisms are defined, and they induce a category of systems. The theory of processes for Local Action Systems is elaborated in [Ver02]; in particular, an equational proof is given for the correctness of the construction enabling one to replace a production by a refinement. It is also demonstrated how this result can be used to construct a compositional semantics for interactive systems by introducing a generalized notion of a production where the left-hand side is a process instead of a graph.

Views and components The idea of a combined reference model- and view-based specification approach has been proposed recently in the software engineering community. In [EHTE97] and [EEHT97], a specification technique based on graph transformations is presented which supports such a development approach. The use of graphs and graph transformations supports an intuitive understanding and an integration of static and dynamic aspects on a well-defined semantical base. On this background, formal notions of view and view relation are developed and the behaviour of views is described by a loose semantics. A construction for automatic view integration is 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 case of more than two views more general scenarios are developed and discussed in [EEHT97].

Graph transformation systems support the formal modeling of dynamic, concurrent, and distributed systems. States are given by their graphical structure, and transitions are modeled by graph transformation rules. In [GSP98b], two kinds of refinement relations for graph transformation systems are investigated in order to support the development of a module concept for graph transformation systems. In a spatial refinement each rule is refined by an amalgamation of rules, in a temporal refinement it is refined by a sequence of rules.

In [GPS98], 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 [GSP98a], the properties of spatial and temporal refinements are further investigated while, in the second part, they are employed for the development of a module concept for typed graph transformation systems. Finally, as a first step towards an algebra of modules, the operations of union and composition of modules are introduced.

In [EEHT98], 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 realtions 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 modlues and interconnections? In [EEHT98], these questions are answered for five different module concepts for graph transformation systems.

In previous papers of Ehrig and Orejas, a conceptual and a formal model for an integration paradigm for data type and process modeling techniques has been presented. This concept leads to a generic concept of integrated model specifications and corresponding models, where layer four of this integration paradigm includes already a basic component concept.

The aim of [EO01] is to present the general version of this formal component concept based on integrated model specifications. This component concept is generic with respect to several parameters, which can be instantiated according to a large variety of integrated data type and process modeling techniques. The component concept provides explicit import and export interfaces. Different notions of composition of components are defined via match of corresponding interfaces. The main results show consistency and compositionality for the semantics of components and their composition. This generalizes corresponding concepts and results for the algebraic module concept and module concepts for graph transformation systems to components for integrated data type and process modeling techniques. These concepts are further developed towards a transformation-based component framework in [EOB+02a, EOB+02b] and especially useful to be instantiated by graph- and net-based techniques. The main concepts are a self-contained semantics and internal correctness of components, based on a new idea of high-level constraints. Two main results concerning compositionality show that semantics and correctness for a system can be inferred from that of its components. The concepts are illustrated by a running example on modeling Java threads by high-level nets.

In [Gru00], 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 [Gru00] how any meta-typed graph grammar can be transformed into equivalent ordinary typed SPO graph grammars.

4.1.2 Parallelism and Concurrency

In the first phase of the project, several approaches to the concurrent semantics of systems have been generalized from Petri nets to GTSs. More specifically, focusing on the so-called *double pushout* (DPO) *algebraic approach* to graph rewriting, we have provided GTSs 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 notion of concatenable graph process is introduced in [BCM98a], which also shows that the process semantics is equivalent to the classical semantics for GTSs based on shift-equivalence (derivation trace semantics). The papers [BCM98c, BCM99] face the problem of defining an unfolding semantics for (DPO) graph transformation systems. A construction for unfolding any graph grammar into an acyclic branching structure, that is itself a (nondeterministic occurrence) graph grammar is presented in [BCM98c], and the correspondence with Winskel's is pushed further in [BCM99], where it is shown that the unfolding can be abstracted naturally to a prime algebraic domain and then to an event structure semantics. Such event structure coincides both with the one defined by Corradini et al. via a comma category construction on the category of concatenable derivation traces, and with the one introduced by Schied, based on a deterministic variant of the DPO approach. These results, besides confirming the appropriateness of the proposed unfolding construction, unify the various event structure semantics for the DPO approach to graph transformation.

It is worth mentioning that a basic role in the work on GTSs has been played by the study of *contextual nets* (also called nets with read, activator or test arcs) and *inhibitor nets* (or nets with inhibitor arcs), two generalizations of Petri nets, already known in the literature, which can be seen as intermediate models between ordinary nets and GTSs. The study of *contextual nets* is carried out in [BCM98b, BCM01], where 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. The second paper introduces also a notion of concatenable (deterministic) process of contextual nets, showing a coherence result between the process and unfolding semantics. This work is generalized to inhibitor nets in [BBCP00]. Due to the non-monotonicity of the enabling in nets with inhibitor arcs, 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.

A comprehensive account of the concurrency theory of DPO graph transformation systems can be found in [BCE$^+$99, Bal00].

According to the so-called 'Petri nets are monoids' approach, the multiset structure of markings can be lifted to the computation level to characterize the concurrent net behaviors algebraically. In [BMMS99, BMMS01] we show that although the algebraic semantics of place/transition Petri nets under the collective token philosophy can be fully explained in terms of strictly symmetric monoidal categories, the analogous construction under the individual token philosophy is not completely satisfactory, because it lacks universality and also functoriality. We introduce the notion of pre-nets to overcome this, obtaining a fully satisfactory categorical treatment, where the operational semantics of nets yields an adjunction. This allows us to present a uniform logical description of net behaviors under both the collective and the individual token philosophies in terms of theories and theory morphisms in Partial Membership Equational Logic [BMMS98, BMMS99]. Moreover, since the universal property of adjunctions guarantees that colimit constructions on nets are preserved in our algebraic models, the resulting semantic framework has good compositional properties [BMMS01].

In [BS00] (full version in [BS01]) the 'Petri nets are monoids' approach is extended to contextual nets by considering monoidal categories of computations over a non-free monoid of places: a token is regarded as an *atom* that can emit several 'negative' particles that can be read (but not consumed) separately.

Dynamic change and evolution in distributed systems Mobile and distributed systems typically consists of several layers: Objects on the lowest layer are clustered by components on the second layer, which themselves are located at nodes of a computer network on the third layer. Orthogonal to the three layers, an instance level and a type or schema level are distinguished when modeling these systems. Accordingly, the changes a system experiences during its lifetime can be classified as the system's dynamic behavior on the instance level and as the evolution of the system on the schema level. In [EH00b, HE01] graph transformation is applied as conceptual and formal framework for describing the structural and behavioral aspects of such systems.

Graph transformation with time Following TER nets, an approach to the modelling of time in high-level Petri nets, [GH02] proposes a model of time within (attributed) graph transformation systems where logical clocks are represented as distinguished node attributes. Corresponding axioms for the time model in TER nets are generalised to graph transformation systems and semantic variations are discussed. They are summarised by a general theorem ensuring the consistency of temporal order and casual dependencies.

The resulting notions of *typed graph transformation with time* specialise the algebraic double-pushout (DPO) approach to typed graph transformation. In particular, the concurrency theory of the DPO approach can be used in the transfer of the basic theory of TER nets.

Local action systems Research concerning concurrency aspects for graph rerwiting with embedding has concentrated on Local Action Systems (LAS), introduced in[Jan99], which generalize actor grammars and ESM systems by removing some ad-hoc features. The research has concentrated the development of a theory of processes, starting from ideas from Petri nets. It has been shown in [Ver02] 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 [JV01] and [Ver02] it is shown that the process theory of Local Action Systems can be instantiated for the special case of Petri Nets. Considering various straightforward encodings of markings by discrete graphs, one obtains in a uniform way several versions of the theory. A number of these correspond to variants of Petri Nets studied in the literature. Thus one may obtains a unified view of processes for these variants, including Place/Transition nets, Elementary Net Systems, Coloured nets and Nets with Inhibitor Arcs.

4.1.3 Semantics, Correctness and Optimisation

The concurrent semantics of GTS defines operational models of computation which give a concrete description of their behaviour in terms of non-effective (e.g., infinite, non-decidable) structures. Thus they cannot be used directly to reason about the modelled system. Indeed, these operational models are intended to represent the basis for the definition of more abstract semantics, which take into account only some aspects of interest for the system at hand, and of effective verification techniques based on such semantics.

Following this line of research, the paper [BCK01] proposes a verification technique for GTSs based on the construction of an approximation of the unfolding semantics. In [BCK02] such technique is used to prove a security property for a distributed system with mobility modelled as a graph transformation system.

Extending the work developed for contextual nets in [BCM00], the paper

[BCM02] proposes an abstract semantics for graph transformation systems based on history preserving bisimulation (HP-bisimulation), a behavioural equivalence sensible to the concurrency properties of the systems. Resorting to HD-automata, a variation of ordinary automata suited to deal with history-dependent formalisms, HP-bisimulation is shown to be decidable for finite-state graph grammars and an algorithm is proposed to decide such equivalence. The paper presents also the logical counterpart of HP-bisimulation, i.e., a logic in the style of Hennessy-Milner adequate for HP-bisimulation.

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

In [HMP00], 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, HMP01], the classical theorems on parallelism and concurrency are adapted to the setting with injective matching.

The traditional double-pushout approach to graph transformation does not allow to change node labels in an arbitrary context. The paper [HP02b] proposes a simple solution to this problem, namely to use rules with partially labelled interface graphs and to match rules injectively. In [HMP01] it was shown that injective matching makes the double-pushout approach more expressive, and here that approach is further generalised. Besides solving the relabelling problem, the proposed framework allows to write rules with partially labelled left-hand sides which are equivalent to (possibly infinite) sets of rules in the traditional setting. Unlike previous work on rules with partially labelled graphs, the approach does not need any labelling condition on matching morphisms, nor does it exclude node merging rules.

Redundancy In [KV00] the notions of redundancy and subsumption – as known from the area of rule-based systems – are considered in the context of system verification. A rule is redundant if it can be removed without affecting the semantics of the system; it is subsumed by another rule if each application of the former one can be replaced by an application of the latter one with the same effect. Redundancy and subsumption are treated in the framework of high-level replacement systems, which generalize graph and hypergraph grammars. The main results of the paper are a characterization of subsumption and a sufficient condition for redundancy, which involves composite productions.

Context-free graph specification The context-free syntax of programming languages like Pascal and Modula is often defined by means of syntax diagrams in a quite intuitive way. The syntactically correct programs are obtained by reading words along paths in syntax diagrams in a certain way. In [DK01b], hyperedge replacement graph languages are interpreted as sets of syntax diagrams, and the string languages definable in this way are investigated.

In [Dre98a, Dre97] the structure of hyperedge-replacement graph languages consisting of (hyper)trees is investigated. It is shown that these languages can be generated by productions whose right-hand sides are themselves hypertrees. However, for this one has to turn to sets of derivation trees produced by a slightly more powerful generation mechanism than usual, namely finite-copying top-down tree transducers. Thus, finite copying compensates exactly the loss in power that results from allowing only hypertrees as right-hand sides.

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

The idea motivating the doctoral thesis [Kle01] is to combine the two most important context-free (hyper)graph grammar approaches, namely confluent node rewriting (C-edNCE rewriting) and hyperedge rewriting (HR rewriting), into a framework which maintains the strengths particular to the original approaches. For this, confluent node rewriting in hypergraphs – C-hNCE rewriting – is established as the most powerful known context-free hypergraph grammar approach; see also [Kle99] and the overview in [Kle02a]. Moreover, it is shown in the thesis as well as in [Kle02b, Kle00] that for the so-called 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. As node-rewriting grammars are not naturally confluent, a decision algorithm is developed. Subsequently, a combination of node rewriting with the well-studied hyperedge-rewriting approach is studied, yielding the concept of atom replacement. These methods to rewrite hypergraphs offer adequate formal frameworks for Petri net refinements. Finally, simulations of context-free hypergraph rewriting (see also [JK00]) as well as Petri net refinement by pullback rewriting (compare [Kle98]) provide a link to category theory. In [BJK02], the basic definitions and results of the pullback approach to graph rewriting are reviewed.

Dynamic meta modeling It is common (e.g., in the meta model of UML) to use graphs for representing the abstract syntax of diagrams. Generalizing term-based techniques for defining the semantics of (textual) programming languages we develop graph-based approaches to the semantics of diagram languages [EH00a, HE00]. In particular, we have generalized Plotkin's structural operational semantics (SOS) to the *graphical operational semantics* (GOS) approach [CHM00]. Denoting GOS rules as UML collaboration diagrams, the approach is used to specify the dynamic semantics of the UML [EHHS00, HS01, HS00, HZ01, HHS01].

4.1.4 Language Design

The paper [HP01] identifies a set of programming constructs ensuring that a programming language based on graph transformation is computationally complete. These constructs are (1) nondeterministic one-step application of a set of graph transformation rules, (2) sequential composition and (3) iteration in the form *apply as long as possible*. This language has a simple formal semantics and is minimal in that omitting either sequential composition or iteration results in a computationally incomplete language. Computational completeness refers to the ability to compute every computable partial function on labelled graphs. The completeness proof is based on graph transformation programs which encode arbitrary graphs as strings, simulate Turing machines on these strings, and decode the resulting strings back into graphs.

The paper [HP02a] shows how to program in the core programming language investigated in [HP01]. The presented programs compute various functions and relations on graphs: the functions generating the transitive closure of a graph and the disjoint union of all subgraphs of a graph, the relation yielding all spanning trees of a graph, and functions testing for connectedness, acyclicity, and planarity.

A paper *On diagrammatic languages and their editors* has been presented at DIAGRAMS 2002 ([BC02b]). The paper deals with a metalevel approach to the definition of families of visual languages. Diagrammatic languages are characterized by the existence of identifiable elements in some spatial relations. By progressively specifying the types of relations and the the types of elements which can be involved in them, subfamilies can be derived up to the definition of concrete syntaxes. The approach lends itself to the systematic derivation of rules or rule schemata to construct correct sentences or to parse them. The structure of this rule is independent of the specific language or family of languages, so both graph transformations, high level replacement, multiset or Chomsky-like grammars can be used. Syntax directed editors and parsers can be automatically constructed from such rewriting systems.

PROGRES The PROGRES language combines two successful principles in one formalism: rules and graphs. This is the common vision of the graph transformation community and the main motivation behind the development of PROGRES, a visual language that supports PROgramming with Graph REwriting Systems (see [SWZ99]). Specification and rapid prototyping of graph manipulation software by using a graph transformation language such as PROGRES has attracted more and more interests in various fields of computer science. Since 1997, the PROGRES language has been extended considerably to develop into a language which is nowadays used in many research projects at many sites world-wide.

The main purpose of the PROGRES language is specifying process modelling tools, database query languages, or visual language environments. Very soon, the typical size of a specification grew to about 100 to 200 printed pages. The lack of a module concept for graph transformations became apparent. First, a Modula-2 like module concept has been developed which was published along with ideas how to use active database system technology for defining abstract graph views in [WS97]. After that, this module concept was refined and adapted a Java-like package concept resembling the object-oriented modelling standard UML. For this purpose, the package concept of the UML has been formalised and improved. Research results were published in [SW97] and [SW98].

Based on this module concept and the research results of the evaluation of this concept an object-oriented extension of PROGRES has been developed, i.e. graph transformations have been added to node class definitions as commonly known from other OO languages such as e.g. C++. This need has shown up quickly after having studied an important application domain of PROGRES, visual programming and modelling languages (see [MS99]).

Research results about the extension to a truly object-oriented language will be published after the expiration of the APPLIGRAPH project in [Mün02].

In [MSW00] a closely related language design topic is presented, the integration of static integrity constraints into the PROGRES language. These constraints check a specification's semantics at run-time. Active integrity constraints which are well-known in the world of active database systems enable the user to use an event-based programming methodology in PROGRES. In conncetion with modules, constraints are a crucial means for programming by contract and for constructing updatable graph views.

In 2001, the polymorphic typing concept of PROGRES has been extended. PROGRES is a statically typed language. The integrated analysing tool is able to validate a PROGRES specification with more than 300 rules context-sensitively. With the help of a two level typing concept, a polymorphic typing system has been implemented. This typing system has been extended by parametric polymorphism. Templates of node classes have been specified which are instantiated later by special parameters, maintaining the ability of static type checking. In [MS00] the concepts and ideas of this research topic have been presented.

Specifying agent-based systems In [DHK01c, DHK01a], typed graph transformation systems are used as a rule-based modeling notation for 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 rules describe local, autonomous operations by which agents may react to changes in their environment. Graph transformation rules are controlled by means of state diagrams which describe the temporal ordering of an agent's local operations.

The approach is formalized in [DH00a]. In [DH00b, DHK00a, DHK00b, DHK01b], a concept of roles is introduced in order to model interaction protocols as well as the evolution of agents' structure and behavior.

OO requirements engineering In object-oriented software development, requirements of different stakeholders are often manifested in use case models which complement the static domain model by dynamic and functional requirements. In the course of development, these requirements are analyzed and integrated to produce a consistent overall requirements specification. Iterations of the model may be triggered by conflicts between requirements of different parties. However, due to the diversity, incompleteness, and informal nature, in particular of functional and dynamic requirements, such conflicts are difficult to find. Formal approaches to requirements engineering, often based on logic, attack these problems, but require highly specialized experts to write and reason about such specifications.

In [HH01] we propose the use of graph transformations to formalize the relation between different aspects, i.e., an integrated model combining UML use case, activity, and collaboration diagrams. The formalization allows to make precise the notions of conflict and dependency between functional requirements expressed by different use cases. Then, use case models can be statically analyzed, and conflicts or dependencies detected by the analysis can be communicated to the modeler by annotating the model. An implementation of the static analysis within a graph transformation tool is presented in [HHT02].

4.2 Tool Issues

The APPLIGRAPH subarea *tool issues* mainly deals with the design and development of graph transformation tools. This research is influenced by both application issues concerning application domains of the realised tools, and language issues concerning the implementability of specification and programming languages. In the following, the tool building activities of all APPLIGRAPH members and their scientific correspondents are listed. Almost all presented projects deal with the development of editors, parsers, and interpreters based on graph transformation technology.

4.2.1 AGG

Started in the beginning of 1997, the development of the graph transformation system AGG (for Algebraic Graph Grammar system) is an on-going activity, putting into practice the algebraic Single-Pushout approach [EHK+97] with several extensions. Although retaining the name AGG from [LB93], the system is redesigned from scratch, merely taking over some ideas and concepts from its predecessor. The implementation is done purely in Java, a modern object oriented, platform independent programming language. The AGG system provides a platform for realization and illustration of theoretical concepts based on the algebraic approach to graph transformation, and it aims to evolve into an environment for visual specification and programming.

[Rud97] gives an in-depth description of the object oriented design and implementation of AGG's core graph transformation machine, with a focus on the problem of graph matching, which is formalized and solved as a constraint satisfaction problem. For a correctness proven implementation of a proper graph transformation step, which is given by the categorical construction of a colimit, Rudolf relies on the colimit library of [Wol98]. [Rud97] also covers the integration of the attribution component developed in [Mel97]. This component allows to annotate the nodes and arcs of a graph with values of arbitrary Java types, which can then be operated on using arbitrary Java methods in graph rules. With this concept, Melamed provides a powerful means for smooth integration of textual and visual programming. In [Mel98], he extends the component to meet the requirements of distributed graph transformation, and adds several concepts like constraints on attributes and parameter passing.

While the aforementioned efforts of Melamed and Rudolf focus on the detailed documentation of AGG's design and development, Taentzer, Rudolf, and Ermel [ERT99] give a survey of its concepts, functionality, and use from the user's perspective. By examining some examples, they show how the "AGG-approach" of integrating conventional textual programming in Java with graph transformation may be employed to solve common programming tasks.The 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 [ERT99], 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 side, the wheel is not reinvented again, since well elaborated programming concepts, tools and class libraries mirroring the state of the art of programming can be used. The 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.

[Rud00] presents a way to represent and solve the problem of graph matching - also known as the subgraph homomorphism problem - as a constraint satisfaction problem (CSP), opening up direct access to the large variety of research findings on optimized solution algorithms for CSPs. By decoupling the solution algorithm from the concrete graph model, the approach allows for variations of the model without affecting the algorithm. Furthermore, complementing the standard CSP definition, a query concept is introduced to allow abstract representation of concrete implementation properties for optimization purposes.

4.2.2 GenGEd

In [NB00] GenGEd is presented. This system is used to interactively specify and generate syntax-directed editors for visual languages. In analogy to textual languages a visual language is specified by both an alphabet and a grammar. Hence, the GenGEd system provides an alphabet and a grammar editor. The grammar rules specify not only language generating productions but additionally the edit commands of the visual language editor. The generated editor can be used in various environments to allow for syntax-based drawing of diagrams.

In [BEE+02], an overview is given about the current concepts of GenGED, an environment for the visual definition of visual languages (VLs). From the visual definition, a VL specification is obtained that serves as a configuration of a VL-specific environment, i.e., the configuration is dependent on the parameters available in a VL specification: GenGED allows for the visual specification of syntax-directed editing, parsing, and simulation as well. In addition to these features, we show how to define an animation view for a certain VL model. All GenGED features are based on the formal concepts of algebraic graph transformation and graphical constraint solving. Hence, we have a well-defined theory which serves as a basis for proper extensions of GenGED.

GenGEd supports mainly the generation of syntax-directed editors. However, syntax-directed editing is not always desired by the user. Therefore in [BST01], GenGEd is extended by parsing facilities which allow for free editing as well. The implementation goes back to graph parsing in AGG which can use critical pair analysis to gain more efficiency.

The GenGEd approach allows for the generic description of visual modeling languages such as different Petri net classes. In [EBE01], GenGEd has been extended in order to support the description and implementation of animation views for different Petri net classes. The animation view of a system modeled as a Petri net consists of a domain-specific layout and an animation according to the firing behavior of the Petri net type. The basic idea is to generate visual behavior and animation rules based on visual syntax rules defining the corresponding Petri net language. For this purpose we propose a view transformation from the classical Petri net layout to the animation layout. The well-known producer/consumer system modeled as an elementary Petri net serves as running example. The animation view of the model in the application domain of a kitchen shows directly the production and consumption of cakes. Finally, the implementation of animation concepts in the GenGEd environment is presented.

4.2.3 Graph Exchange Formats

The development of common exchange formats for graphs and graph transformation systems is an ongoing initiative within the EU Working Group APPLIGRAPH. [Tae01] reports on the current state of this format discussion. The formats are based on the extensible markup language XML developed to interchange documents of arbitrary types. Graphs are basic structures in various areas of computer science. A common format for graphs supports the interaction of software developers building tools with respect to graph layout, graph algorithms, graph transformation, reengineering, etc. Moreover, within the graph transformation community tool builders can even gain from a common exchange format for graph transformation systems.

4.2.4 Interpreters, Compilers, and Optimisers

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

4.2.5 DiaGen

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 modelled 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.

For rule-based programming with graphs, graph transformation has to be extended by some further concepts. Concepts like abstraction, control, and encapsulation have been studied in [Hof00, HM00d].

With DiaGen, programming with graphs can be extended to *programming with diagrams*, using the visual notation of the program's application domain [HM00e, HM00b, HM00c]. For instance, Statecharts can be edited with an editor generated by DiaGen, and can then be animated or transformed by rule-based programming of their internal representations as hierarchical graphs [MH01b].

Work on DiaGen has been continued into three directions during the APPLIGRAPH research project:

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

Nowadays, most research topics have been concluded. Automatic layout is needed when the user modifies some diagram components which requires other components to be modified as well such that the meaning of the diagram is not altered. Automatic layout can also take care of "beautifying" diagrams which have been created by the user sloppily [CMP99]. Also, automatic layout is needed when providing *syntax-directed editing*. Syntax-directed editing operations are specified by hypergraph transformations which are applied to the internal hypergraph of the diagram [KM00]. In DiaGen, automatic layout mechanism are specified in terms of constraints on the diagram components' attributes. These constraints are attached to productions of the hypergraph grammar which specifies the syntax of the diagram class. A constraint solver is used to solve the corresponding constraint satisfaction problem and to determine the diagram layout.

DiaGen is now completely implemented in Java 2. It consists of a framework for diagram editors and a program generator. The user has to provide a textual specification of the diagram language which 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 [BTMS99]. DiaGen uses external constraint solvers. Currently, Qoca [MCF98] (Monash University, Australia) and Parcon [Gri96] (University of Paderborn, Germany) are used.

4.2.6 The Fujaba Tool Set

During the APPLIGRAPH research project our scientific correspondents from University of Paderborn, Germany, conducted research on combining 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 (From UML to Java And Back Again) environment provides code-generation facilities that allow to generate Java code from class diagrams and story diagrams [NZ00]. This Java code does not rely on a heavy-weight library or database but only on some standard container classes. Thus, the generated code can easily be integrated with other system parts, e.g. a graphical user interface.

In [FNTZ00] we describe a methodological use of story diagrams for the development of object-oriented applications. This starts with the refinement of use-cases via so-called story boards. Story boards have a semantics close to story diagrams. However, while story diagrams are executable formal specifications that allow a direct translation to an object-oriented programming language, story boards describe only possible scenarios. Thus, story boards are used to outline first ideas about the internal behavior of the desired system. In the next step the story boards are analysed and serve as a basis for the formal specification of the behavior of the desired system using story diagrams. In the final phase the Fujaba environment generates code from the story diagrams that is comined with a graphical user interface and the other system parts.

4.2.7 The PROGRES Tool Set

The PROGRES system is an integrated visual programming environment to support the definition and implementation of graph rewriting rules. Our main tasks are the maintenance and development of the PROGRES language and improving the usability of the PROGRES environment. In the past years PROGRES has been used in many research projects, e.g. for process modelling and management, workflow support, specifying CIM environments etc. versioning control and configuration management tools. The functionality of these systems was specified using PROGRES and code was generated automatically by a built-in compiler [SWZ96]. Based on this code prototypes have been created utilising the UPGRADE framework.

The Aachen research group collaborates with a number of partners from industry such as Springer Verlag, Ericsson, Bayer AG, GEZ, Aachen Münchener Informatik etc. We maintain contacts to user communities within Germany and also world-wide (Canada, UK, France). Berlin, Erlangen-Nürnberg, Paderborn (all Germany), Kingston (Canada), Carleton University Ottawa (Canada), University of York (UK), INRIA research institute (France) etc.

Application domains of PROGRES are studied e.g. in [Jäg00], describing a PROGRES-based approach for generating graphical, structure-oriented software engineering tools from graph-based specifications. Another application domain is presented in [Rad00b, 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 [Rad00c] presents a tool that prepares monolithic software applications for distribution by applying a sequence of transformations. 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. Reverse engineering techniques using PROGRES are also investigated in telecommunication industry [HM01a]. PROGRES is used to develop methods, concepts, and tools to support the processes of understanding and restructuring complex legacy telecommunication systems.

The PROGRES tool set combines two successful principles, 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 [Mün00b]. 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.

In the past years PROGRES has been extended by several concepts allowing for an easier way of specifying graph grammars and the functionality of rapid prototypes. Throughout the years, PROGRES' user community has grown and the problems tackled with this system have become more and more complex. Consequently, specifications grew larger and larger. The demand for structuring means as known from programming in the large became evident. Therefore, a UML-like module concept was integrated for which UML's semantics had to be precisely defined [SW00, SW97, SW98] as well as means for checking a specification's run-time semantics [MSW00].

Research activities also comprised the improvement of the PROGRES language and environment. The integration of parametric polymorphic typing concepts to the PROGRES language was a further step [MS00]. With polymorphic typing concepts the users are able to specify generic graph class templates. The core research of the PROGRES group regarding application of graph transformation dealt with the definition of generic modules for editing, analysing, and interpreting visual languages. With PROGRES, a set of reusable modules for a visual standardised language which is heavily used in automation industry, the IEC-61131/3, was developed. These modules can also be reused for the definition of other component-based visual languages. Also, the opportunity to define a behaviour of graph elements to the already existing object-based specification methodology, resulting in an object-oriented specification approach was integrated.

4.2.8 Further Tool Development Activities

The system TREEBAG implements the idea of tree-based object generation and in particular picture generation. It is presented in [Dre98c, Dre98e, DK00c, DK01a].

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

In this subarea, the intention is to demonstrate the usefulness of graph transformation by case studies in various fields like definition of visual languages, database models, concurrent and distributed systems and software process modelling.

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

Visual techniques are important for specification, modelling and programming (visual computing) and they can be used to support component technology, evolution and the software development process (visual development of software components). [ET00a] points out the central role of graphs and graph transformation for visual computing and visual development of software components. Moreover, it discusses the need for consistent integration of both areas and to propose graph transformation as a suitable technique for this purpose. The general idea of this position paper is not to present new technical results but to propose new lines of research how to combine and extend current projects and proposals in the areas of visual computing and visual development of software respectively.

The technical report [Bar97] is a summary of an analysis of visual languages starting at the end of 1996. The reason for this analysis is the development of a visual (programming) environment based on algebraic specification techniques and graph grammars. In the context of software engineering this analysis can be understood as an informal requirement specification. It discusses some application areas of visual languages in general and diagram editors in particular. Furthermore, the report treats some approaches with a similar aim, namely the development of visual environments based on graph grammars. It ends with an informal requirement specification corresponding to how to represent a diagram, and some conceptual ideas for tool support. Based on the ideas mentioned in [Bar97] the concepts for a visual environment are presented in [BT97]. The main and very new feature is the graphical definition of visual languages as the basis to generate supporting editors. Roughly, a user has to determine the visual means of a visual language and links between these means leading to an algebraic specification. On top of the algebraic specification graph grammar rules are used to define visual languages rules. These rules define the edit commands of a syntax directed editor.

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 considered carefully against efficiency of tools, e.g., when parsing visual languages. Context-free as well as different kinds of context-sensitive languages are considered in [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 the application of graph transformation rules. General purpose languages PROGRES and AGG as well as special purpose languages DiaGen and GenGEd are discussed and compared.

The GenGED approach The overall aim of the GenGED approach [Bar98, Bar00, Bar02] is the visual definition of visual languages which can then be used 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 the generators for visual environments are available. 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 visual environment GenGEd supports the visual definition of visual languages (VLs). Each VL is defined by an alphabet and a grammar. From a specific VL-definition, a VL-specification is generated which is the input of a graphical editor allowing for syntax-directed editing of diagrams over the specified VL. GenGEd as well as each VL is based on the well-defined concepts of algebraic graph transformation and graphical constraint solving. The underlying formalism is hidden from the user, but it is essential for a formal presentation and manipulation of graphical structures. In [Bar02], the main concepts of GenGEd and the underlying tool are illustrated by the definition of a simple kind of the well-known statechart language. [BE00] presents the conceptual model of GenGEd which is based on algebraic graph transformation and algebraic specification techniques. [BK98] discusses the use of GenGEd to construct an editor for a kind of graph grammars that can be used to specify reactive systems. [BNS00] concentrates especially on the visual tool environment. The GenGEd environment offers a hybrid language for defining the syntax of visual languages. The syntax description is the input of a diagram editor allowing the syntax-directed manipulation of diagrams.

Simulation of models by GenGED The use of simulation within software development has as main aims to validate the system, as well as to recognize performance aspects. A simulation model includes the description of the behavior of the systems, but also intended animation strategies (to visualize the simulation) and statistics collecting/evaluating procedures. In [BER00], a modular approach for animating simulation models is presented. Both the system's behavior and its animation are modeled by graph grammars. The simulation of the system as well as its animation are supported by tools. We discuss the integration of the tools PLATUS, designed to construct the simulation models, and GenGEd, designed for the development of visual languages, in order to obtain an environment in which animation modules can be obtained (from GenGEd) and used (in PLATUS). Moreover, due to the modular structure, an animation can be exchanged easily, and can be also reused by other simulation models. The Platus simulation environment allows for a modular description of simulation models, clearly separating aspects of behavior, statistics and animation. Each component of a model in Platus has its own animation interface, that describes when and which messages will be sent to a corresponding animation module. In [BR01], GenGEd is extended in order to define several animation modules which can be connected via animation interfaces to Platus modules, allowing for a visual animation of the application being simulated.

Application of GenGED to behavioural models and animation In [BEE01], the description of dynamic behavior and animation of systems is presented. The basic idea is to define visual behavior and animation rules on top of the rules defining the corresponding visual modeling language and to allow a domain specific layout for the animation view of the system. Two visual language models based on the visual language of Petri nets serve as running examples: a simple version of a traffic light system and the well-known specification of The Dining Philosophers. The system views are given by means of high-level Petri nets. The animation view of the traffic light system shows directly the dynamic changes of the colors of traffic lights at a street crossing, whereas the animation view of The Dining Philosophers visualizes the behavior of philosophers sitting around a table.

Visual language parsing [BST00a, BST00b] propose an extension of layered graph grammars (LGGs), which have been introduced for the definition of visual languages (VLs). Offering new constructs like negative application conditions (NACs) allows for more concise VL definitions. A new layering condition and critical pair analysis are the prerequisites for a new parsing algorithm for visual languages which avoids the exponential behaviour of LGGs in many cases. The new type of graph grammar is applied to the definition of a simplified form of Statecharts.

Formal aspects of visual language definition In a rule–based approach the computation steps of a system are specified by rules that completely define how the system's state may change. For open systems a more liberal approach is required, where the state changes are only partly specified, and—interactively—other components may contribute further information on how the transformation is defined completely. In [BGRS01], a formal model for interactive rule–based specifications is introduced, where states are modelled as partial algebras and transformations are given by internal algebra rewritings and arbitrary external components. As an application we discuss how visual languages can be defined in this framework. Thereby the internal (logical) representations of visual expressions are transformed by rewriting rules, whereas their layouts are obtained interactively by external components like a constraint solver or a user working with a display and a mouse.

[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.

Definition of the statechart language On the Symposium on Visual Languages and Formal Methods being part of the IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01) there was a special statechart modeling contest. In [BT01] and [BE01], a slightly simplified version of statecharts as they are supported by the CASE tool Rhapsody is considered. In [BT01], contextual layered graph transformation as introduced in [BST01] is used to define high-level parsing rules and the operational semantics for this kind of statecharts. In [BE01], GenGED is used for the visual specification of both, syntax-directed editing commands of a language-specific graphical editor as well as for the definition of parsing rules based on contextual layered graph transformation.

Visualization of OCL In [BKPT01], a visualization of OCL is proposed within the context of the UML meta model, so that OCL expressions are represented by extending collaboration diagrams. The description of properties of objects is based on collaborations, where classifier and association roles are used to describe navigation paths. Operations computing properties are described by interactions consisting of messages between classifier roles. The introduction of new graphical core elements is kept to a minimum. New notation mainly concerns the predefined operations in OCL and provides more convenient visual forms for the notation by interactions here. The proposed visualization is described in detail and is illustrated with examples taken from an industrial project under development. Main ideas for this kind of visualization are derived from a graph-transformation based approach. Moreover in [BKPT00, BKPT02], we discuss how an operational semantics for OCL can be based on graph transformation units which guide the application of graph-transformation rules.

Visual languages are often used in software development for modeling databases, software components, and the like. A particular application area, that has emerged in the last decade, is that of hypermedia application design. A hypermedia application usually provides information about a specific domain, and a navigation structure (hyperweb) that allows to browse through it. Both the underlying information and its navigation structure are normally designed using conceptual models based on visual languages (see e.g. RMM by Isakowitz Stohr and Balasubramanian, OOHDM by Schwabe, Rossi and Barbosa). While these languages provide very advanced features for modeling static aspects, they lack primitives for modeling transformations on a hyperweb and on its underlying information base. In [BtH00], an extension to the RMM model is proposed, where graph transformation systems are used for specifying such transformations. Future extensions concern, among others, the use of hierarchical graphs, as suggested by the OOHDM model.

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

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

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.

Collage grammars are an important means to specify visual languages. While a 2-dimensional grid picture grammar may generate pictures (defined as subsets of the unit square) with arbitrarily small details, only a finite number of them can be made visible as raster images for any given raster. In [DEKK00a, DEKK00b] and in [DEKK01], an algorithm based on bottom-up tree automata is presented which computes the set of all raster images of the pictures generated by a given grid picture grammar.

From the perspective of computer science, interlacing knotwork, which forms a significant part of celtic art, is a visual language following mathematically precise rules of construction. In [DK00a, DK00b], the syntactic generation of celtic knots with collage grammars is studied. Several syntactic regulation mechanisms are employed in order to ensure that only consistent designs are generated. Practical research for the papers summarised above was carried out with the help of the system Treebag, which is briefly presented in [DK01a].

Tree-based picture generation In [Dre98d, Dre00b] the concept of tree-based picture generation is introduced. This yields a unified framework for the generation of pictures that can be extended to other sorts of objects as well. In the paper it is shown that collage grammars, iterated function systems, chain-code grammars, and 0L-systems with turtle interpretation are equivalent to natural types of tree-based picture generators.

Context-free collage and chain-code grammars are compared with respect to their generative power in [Dre98b, Dre00a]. It is shown that context-free collage grammars based on similarity transformations can be simulated by context-free chain-code grammars, but that, in general, the language classes generated by both types of grammars are incomparable.

In [Dre99, Dre01] 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 [DK00c].

4.3.2 Database Models and Information Systems

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

The activities within a computer system can be viewed as a sequence of operations on objects. One of the primary purposes of security mechanisms is *access control* which consists of determining and enforcing which active entities, e.g processes, can have access to which objects and in which access mode.

In [KMP00] we have presented a formalization of the Role-Based Access Control models using graph transformations. The advantages of this formalism include the presentation of a formal framework which allows the specification and verification of consistency requirements, and a comparison of several possible solutions to problems, such as the revocation cascade of users membership when administrative roles are decentralized. The comparison of different Access Control policies within the graph transformation formalism and the analysis of the system behavior in combining different Access Control policies (such as a discretionary AC and a mandatory AC) are the subject of [KMP01a]. A policy framework is defined as a 4-tuple consisting of a type graph, a set of graph rules (partial graph morphisms) and two sets of graphical constraints describing what is forbidden and what is required in the state graphs derivable with the rules. In [KMP01b] we have addressed the problem of combining the graph based specifications of two access control policies (even if based on different underlying access control models) while in [KMP02] we have studied how to detect and resolve conflicts between two rules, between two constraints, and between a rule and a constraint, deriving from the integration of two policies.

4.3.3 Concurrent and Distributed Systems

Volume 3 of the *Handbook on Graph Grammars and Computing by Graph Transformation* [EKMR99] gives a comprehensive and state-of-the-art overview on the main work in this field concerning parallelism, concurrency, distribution and related topics. Exactly these topics are central issues of computer science, and their importance will increase in the future. The following topics are investigated in [EKMR99]: graph relabelling systems and distributed algorithms, actor grammars and local actions, concurrent semantics of algebraic graph transformation, modeling concurrent, mobile and coordinated systems via graph transformation, distributed graph transformation with application to visual design of distributed systems, high-level replacement systems applied to algebraic specification and Petri nets and describing systems of processes by means of high-level replacement. The application-oriented contributions are described in more detail below.

Distributed graph transformation The new approach of distributed graphs and graph transformation as developed in [Tae99b] allows to use structured graph transformation on two abstraction levels, the network and the local level. The network level contains the description of the topological structures of a system. The local level covers the description of states and their transitions in local systems. Local state transitions may depend on others using suitable synchronization mechanisms. The main distribution concepts of categorical graph grammars presented by Schneider are combined with the algebraic approach to distributed graph transformation introduced by Ehrig et.al. Modeling of distributed systems by this new kind of distributed graph transformation offers a clear and elegant description of dynamic networks, distributed actions as well as communication and synchronization using a graphical notation. Moreover, distributed graph transformation offers the possibility to describe splitting and joining of local graphs as well as parallel transformations in local systems. The formalization of distributed graph transformation is done by means of category theory. A distributed graph is formalized by a diagram in the category GRAPH of graphs and total graph morphisms. A distributed transformation step is characterized by a double-pushout in the category DISTR(GRAPH) of distributed graphs and distributed graph morphisms. A pushout over distributed graph morphisms cannot always be constructed componentwise in each local part. But especially the componentwise construction of a distributed transformation reflects distributed computations best. Thus, we present the necessary conditions for componentwise construction of pushouts in DISTR(GRAPH) and show that they are also sufficient. These conditions are summarized in the distributed gluing condition. Moreover, these conditions are needed for componentwise construction of pushout complements which are used to characterize the first step of a distributed graph transformation.

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 [GPST00] 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 with respect to 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.

In [Koc97a] distributed graph transformations are extended by distributed application conditions needed for specifying transaction models for distributed systems suitable. The concepts developed are applied to distributed transactions with two-phase commit.

In [TK97] distributed graph transformations are extended by attributes. The approach is general enough to treat different attribution concepts already developed for non distributed graph transformations. The formalization is explicitly done for one of this approaches that seems especially suitable for the practice, in particular the implementation.

In [FKTV99] 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.

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.

A single-pushout approach to distributed graph transformation is developed in [Koc99] that specifies the topology of the distributed system as well as its local data states. 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 [GR00].

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.

For the illustration of the concepts [FKTV99] presents a reference application modeling distributed version control for any kind of document. This subject becomes the more important the more remote software development becomes popular. In contrast to server-client solutions where all documents are held centrally, the documents are really distributed over the involved sites in our case.

In [Koc97b] graph transformations is used to specify transcation models. ACID transactions are used as an example for this specification and it is shown how the property of isolation of these transactions can be checked by a constructive procedure developed in the field of graph transformations.

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

In [TGM00], 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 la 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. We first give an overview of our model of change and how exactly the two aspects of change interact. We describe distributed graph transformation as a technique to realize our change model. An example - a ring database - then shows how our approach can be applied to a small but nontrivial distributed system. This example shows nicely how the two aspects of change can be described uniformly using graph transformation rules and how the interaction of the two change aspects can be defined in an adequate way.

[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.

Distributed computing Distributed object computing is a computing paradigm that allows objects to be distributed over a heterogeneous network. Infrastructures help to develop distributed object applications by offering necessary services for distributed computing. Having a comprehensive infrastructure to hand, the development of complex distributed object systems is feasible in principle. Flexibly evolving architectures as well as highly dynamic distributed object structures are key requirements for nowadays distributed solutions. They can hardly be well designed on this level of programming, due to their complexity. In [Tae02b], the graphical description of the network and its dynamic reconfiguration as well as the component interfaces and local object systems and their behavior is shown. Typical issues in distributed systems like remote object interaction, object migration and replication, communication and synchronization are expressible in this technique. The notation is close to UML. It extends the UML notation where needed. In [Tae02a], a visual framework is presented which offers a more abstract and intuitive approach to the relevant aspects of a distributed object system. In this framework, the UML-like notation, its semantics based on graphs and their transformation, and more abstract on I/O-automata, are distinguished.

Distributed algorithms and dynamic distributed data structures are basic aspects of distributed systems which usually are modelled separately by different specification techniques. In the running example of the mutual access to file systems, typical problems are shown which are not handled by separate modelling. [TE00] discusses how these problems can be solved by integrated specification of distributed algorithms and dynamic data structures by graph transformation. From the component point of view it is even more adequate to use distributed graph transformation for the semantics of distributed system specification.

Graph transition systems In order to account for the specific aspects of reactive and concurrent systems, graph transitions have been introduced as a variant of double-pushout graph transformation. Classical graph transformation systems come equipped with an implicit frame condition which ensures that nothing changed during the transformation beyond what is explicitly specified. They are not suitable for modeling interaction with the environment because they assume complete control about the system and its behaviour. Graph transitions instead provide a more loose interpretation of graph transformation rules. A graph transition preserves, deletes and adds at least what is speicifed in the rule, but it allows additional changes which are interpreted as effects of operations of the environment. A truly concurrent semantics provides an explicit representation of parallelism. In [HLEO00] important results for a theory of concurrency in graph transition systems are presented and shown that they are highly relevant for the specification of reactive systems.

Net transformations Since the application of the general theory of high-level replacement systems has proven to be most rewarding in many different areas, especially in Petri nets, the extension of high-level replacement systems to refinement morphisms is applied to place/transition nets in [PGH00]. The combination of morphisms, that preserve safety properties, with transformations of place/transition nets leads to rules and transformations, that preserve safety properties. Moreover, we extend our approach so that rules can introduce new invariant formulas, that is new safety properties.

The main aim of [BEH+02] and [PU02] is to give an overview concerning applications of graph transformation techniques to the area of Petri nets achieved by the team of TU Berlin within the APPLIGRAPH Working Group and the DFG Researcher Group on Petri Net Technology. In [BEH+02], net transformation systems are considered as instatiation of high-level replacement systems, property preserving net transformations module and component concepts and the modelling of open systems.

[PU02] provides a thorough survey of the work on rule-based refinement. Rule-based refinement comprises the transformation of Petri nets using rules while preserving certain system properties. Petri net rules and transformations are expressed by morphisms and pushouts. This allows an abstract formulation of our notions independent of a specific Petri net class, as place/transition nets, elementary nets, predicate/transition nets etc. Hence, it is adequate to consider our approach as rule-based refinement of Petri nets in general. We have presented various results in recent years at different conferences. So this contribution gives an overview of our work in a compact form leaving out the technical details.

Specification and verification of software architecture 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.

A related approach is proposed in [HM00a], 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.

Paper [HIM00] presents an approach for the specification of software architecture styles using hyperedge replacement systems and for their dynamic reconfiguration using constraint solving. The construction and dynamic evolution of the style are represented as context-free productions and graph rewriting. Reconfigurations are modeled by allowing the declaration, creation and matching of new nodes (i.e. ports of communication) and synchronization is achieved using constraint solving over the productions of the style grammar. A related approach is presented in [HM01b], where synchronization is modeled directly in the graph transformation formalism, by using *Synchronized Hyperedge Replacement Systems*. The capability of creation and sharing of ports together with multiple simultaneous synchronizations provides a very powerful tool to specify more complex evolutions, reconfiguring multiple components by identifying specific ports.

In [KM01] a notion of bisimulation for graph rewriting systems is introduced, allowing one to prove observational equivalence for dynamically evolving graphs and networks. This notion is applied to the framework of synchronized graph rewriting with mobility which is describe in two different, but operationally equivalent ways: on graphs defined as syntactic judgements and by using tile logic. One of the main results of the paper says that bisimilarity for synchronized graph rewriting is a congruence whenever the rewriting rules satisfy the basic source property.

In [FMT01] a simple labelled transition system semantics of Cardelli and Gordon's Ambient calculus is presented, by exploiting a general and flexible model based on (hyper)graphs, where graph transformation is obtained via (hyper)edge replacement and local synchronization with mobility. In addition to tree-like ambients, proposed calculus works just as well with graph-like ambients, which are a more realistic model of internetworks.

Research about the concurrency aspects of graph rewriting with embedding has been concentrated on three issues : the further development of the process semantics of these systems, the use of this semantics for the compositional verification of algorithms, and the relationship with Petri nets. The process semantics of ESM systems (which generalizes the one for Actor grammars) has been refined and generalized to Local Action Systems (LAS), [Jan99].

In contrast to process notions for, e.g., Petri nets the definition of a LAS process does not require the specification of a particular set of productions. 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 with respect to that specification. It has been investigated to what extent this approach can be used for the verification of distributed algorithms; in [Ver02] a distributed shortest path algorithm and a simple transport protocol are considered.

In [JV01] and [Ver02] The relationship between Local Action Systems and Petri Nets is investigated. Apart from the aspects reported in section 1.2, the framework obtained by specializing Local Action Systems to the rewriting of discrete graphs suggests possible extensions of Petri nets, improving their expressive power. As an example, in [Ver02] an extension is considered where dynamic changes of the net can be taken into account.

PROGRES Throughout the years, the graph transformation language PROGRES has been used as a formal specification language and tool in many application areas. The first steps for using PROGRES as specification tool for modelling visual languages have been published in [Sch97]. Using the specification of the new visual dataflow programming language BDL as input, a rapid prototype of a programming environment for BDL was generated. This approach was taken further in [Mün00a], finally leading to the specification of a programming environment for a IEC-61131/3 compliant language, a programming language which is widely used in chemical process engineering and automation industry. Building a fully fledged programming environment for this standard language is still ongoing work at the Aachen University of Technology research site.

Other projects using PROGRES have been developing new concepts for the a posteriori integration of chemical process engineering tools. One of its central components is an administration system which supports simultaneous as well as concurrent engineering activities. PROGRES has been used for the specification of dynamic task nets (DYNAMITE), which may be modelled quite naturally as attributed graphs. Moreover, all operations on these nets, including editing, analysing, and execution are specified by a programmed graph rewriting system (see [HKWJ97, HKW97, Kra98, KKSW00]).

Research results of this collaborative research project formed the basis for a couple of further studies. In [Wes99b] a general approach for models and tools for managing development processes is presented, while [Wes99a] focusses on graph-based product and process management in mechanical engineering. In [BJSW01] our research results are presented which deal with a model for distributes software process management.

A further development of DYNAMITE is described in [JSW00]. 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 (see also [Sch02]). 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 [JSW99, Sch00].

While the development of process modeling tools is one application area for graph transformations of particular interest, another one — the world wide web — might even represent the killer application for graph transformations, we are all looking for. The paper [BGM99a] discusses the perspectives of applying the graph transformation approach of PROGRES [Mün00b] 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).

Another project which works with PROGRES deals with author support : in today's publication process authors transform the ideas they wish to transport to the reader into linear text. The publisher then invests substantial manual labor to add semantical markup and bibliographical data to typeset and structure the document. The structured document is then printed and distributed to the reader who has to infer the intentions of the author from the printed text. We suggest graph-based tools for modeling the intended semantical structure of a text during document construction. The author uses these tools to create the semantical model stored as a graph (cf. [Gat00, GM00b]).

In distributed object-oriented systems, middleware products like CORBA are employed to perform the actual distribution on a network of computers. Existing programs use object-based abstractions, but cannot directly be distributed in the form of a CORBA system, e.g. because not all data types have corresponding IDL definitions. We have developed a tool which supports the distribution process of an existing program. Such a program is modelled as an attributed graph, the needed program transformations from a monolithic program to the required distributed version are implemented as PROGRES graph transformations (see also [Rad00a, Cre99]).

Furthermore, a tool prototype was developed which supports the migration of legacy applications into a distributed environment. As first step of a migration process we have to reverse engineer an application to get a description on a higher level of abstraction. The acquired information is stored in form of graphs (call graphs, control flow etc.) The necessary structural changes are performed on the reverse engineering graphs with the help of PROGRES graph transformation rules [Cre00].

4.3.4 Computer-Supported Cooperative Work

Several contributions to the semantic foundation of Computer-Supported Cooperative Work have appeared whose formal basis is Tile Logic [GM00a]. Tiles are rewrite rules with side-effects, and they are well-suited for modeling coordination languages, since they can be composed (sequentially and concurrently) both in space and in time via possibly complex synchronization and work-flow mechanisms. In fact, tiles exploit a three-dimensional view of concurrent systems: the horizontal dimension (space) is devoted to the modeling of states and components; the vertical dimension (time) models computation steps; and the third dimension (concurrency) accounts for the distribution of activities and resources. This separation of concerns makes it possible to select different flavors of Tile Logic simply by fine-tuning the algebraic structure of the elements in each of the three dimensions and by fixing their interplay. By analogy with Rewriting Logic, the tile model has purely logical presentation, where tiles are considered as suitable sequents subject to simple inference rules.

The categorical foundations of several tile formats whose configurations and observations have analogous additional structure (e.g. symmetries, cartesianity) have been investigated in [BMM98c, BMM02], in terms of the original notion of symmetric monoidal and cartesian double categories. In [BMM98c], two classes of Tile Logic are considered, called Process Tile Logic (PTL) and Term Tile Logic (TTL), where the horizontal structure and the vertical structure consist of (a suitable class of) acyclic graphs in the case of PTL and of ordinary terms in the case of TTL. Categorical models based on double categories are then defined for both PTL and TTL (details in [BMM02]).

Ad hoc extensions of PTL are developed in [FM97a, FM97b, FM00, MT98]. In [FM97a], a tile-based bisimilarity semantics for the asynchronous π-calculus of Honda and Tokoro is presented, and it is proved equivalent to the ordinary semantics. In [FM97b], a tile-based semantics for located CCS is proposed, where the operational and abstract concurrent semantics described in the literature now descend from general uniform notions. In [FM00], the results of [FM97a, FM97b] are fully presented and discussed to outline a general approach to the analysis of process calculi, while the paper [MT98] points out similarities and differences between actors and asynchronous π-agents, exploiting Tile Logic as the embedding framework for the two paradigms.

In [BM02], Tile Logic is exploited to combine different kind of graph *connectors* in space and time, accounting for graph-like representations of resource/location sharing and causal dependencies and encompassing the ad hoc proposals of [FM00, MT98]. The result is the formalization of a meta-theoretic approach to the definition of operational and abstract concurrent semantics for systems with name handling. A detailed case study shows that the straightforward application of the general framework to basic parallel processes yields a concurrent operational semantics whose observations exactly characterize the ordinary causal weak bisimilarity based on Darondeau and Degano's causal trees.

Since several existing languages support Rewriting Logic, the overall aim of [MM98, BMM98c] is to define an implementation of Tile Logic in Rewriting Logic, thus making tile specifications executable. Due to the great amount of issues involved, many results are separately presented. In [MM98, BMM98c], to prove the correctness of the implementation, the relationship between the categorical models of Tile Logic (as fully presented in [BMM02]) and of Rewriting Logic is formalized using Partial Membership Equational Logic techniques. In [BMM98b] some useful rewriting strategies (e.g., discarding computations not allowed in the tile semantics) are defined to make the implementation effective. In [BMM98a, BMM99] the methodology is applied to several case studies (e.g., finite and full CCS). The second and third parts of the PhD Thesis [Bru99] give a detailed accounts of most of the above results about Tile Logic.

In comparison with SOS formats, Tile Logic offers a straightforward extension of observational equivalence for *open systems*, which is finer and more elegant than the one obtained by closure under all ground substitutions. This point is made clear in [BdFMM00a] (full version in [BdFMM00b]), where, by analogy with the theory of SOS formats, the meta-theory of Tile Logic is investigated defining several formats guaranteeing that tile bisimilarity is a congruence.

When configurations and observations come equipped with the same basic algebraic operations, Tile Logic is also suitable for modeling *open ended systems*, where states can be reconfigured at run-time. Dynamic reconfiguration is modeled by observing suitable context embeddings in the operational semantics. In particular, the approach can be stretched to deal with structural axioms on configurations, still guaranteeing the important property that 'tile bisimilarity is a congruence' [BMS00].

In [BMR01], Tile Logic is applied to Logic Programming by exploiting a freely generated double category of pullbacks to model unification and to define the coordination rules between goals and concurrent refutations. The outcome of [BMR01] is the view of Logic Programming as an interactive system, equipped with concurrent and coordinated abstract semantics consistent with the most common semantics of Logic Programming. Moreover, the compositionality of the tile semantics is guaranteed by standard results, as it reduces to check that the tile systems associated to logic programs enjoy the *tile decomposition property*. An extension of the approach for handling constraint systems is also discussed.

In [BM99a], the higher-order extension of Tile Logic is discussed, which can provide an expressive type framework with applications to the area of reactive calculi with name passing, mobility and higher-order features. Correspondingly, categorical models of higher order Tile Logic are defined by means of the original notion of cartesian closed double category. The case study of the π-calculus, where the tile framework straightforwardly handles name passing and creation, is studied in detail. The paper [BHLM01] compares the higher-order version of Tile Logic with the Edinburgh Logical Frameworks, discussing the main features of the two approaches as *meta-languages* and *(computational) meta-models*, and in particular with their use as logic and semantic frameworks for the definition and analysis of languages, calculi and programming paradigms. The comparison traces differences and analogies between the two approaches on the basis of two case studies: late π-calculus and lazy simply typed λ-calculus.

The abstract [BM00b] surveys the theoretical foundation and applications of Tile Logic.

The simplest class of tiles, where both configurations and observations are multisets of basic elements, has been shown to provide Petri nets with the notion of concurrent transaction. The resulting tile models are called *zero-safe nets* and have been investigated in [BM97, BM98, BM00c, BM99b, BM00a, BM01a, BM01b], together with the application to several case studies.

In [BM97, BM98] zero-safe nets are first proposed as an extension of ordinary P/T Petri nets, where a primitive notion of transition synchronization is introduced: besides ordinary places, called *stable places*, zero-safe nets come equipped with *zero places*, which are empty in any stable marking; a transaction is a concurrent computation which may use zero tokens as triggers, but defines an evolution between stable markings only. Hence, the abstract view of a zero-safe net Z is an ordinary P/T Petri net whose places are the stable places of Z and whose transitions are the basic transactions of Z. Results relating zero-safe nets and their abstract counterparts via categorical adjunctions are presented for both the collective and the individual token philosophy.

The theoretical foundation of zero-safe nets is fully investigated in [BM00c] that summarizes and extends the results in [BM97, BM98] with application to several case studies. A tutorial introduction to the subject has also appeared in [BM99b], while the first part of the PhD Thesis [Bru99] is centered around the results in [BM00c].

In [BM00a] a distributed interpreter for zero-safe nets is defined, which is reminiscent of the ordinary net unfolding, while in [BM01a] the approach has been extended to deal with *read arcs* (which allow for multiple access in reading to the same token).

The paper [BM01b] discusses the conditions under which *inhibitor arcs* can be introduced in zero-safe nets without compromising the abstract view of the system and applies *contextual zero-safe nets* to a transactional extension of the coordination language Linda for defining a concurrent operational semantics.

The abstract [BM00d] surveys the theoretical foundation and applications of zero-safe nets.

4.3.5 Design and Implementation of Programming Languages

The operational semantics of languages integrating functional and logic programming are usually based on narrowing which combines rewriting with unification. In practice, narrowing is often implemented on graph-like data structures for efficiency reasons. The completeness of such implementations is studied in [HP97, HP99]. Completeness results for three narrowing strategies are established: minimally collapsing narrowing, maximally collapsing narrowing and basic narrowing. The result on basic narrowing improves and corrects a recent result of Krishna Rao (published in *Proc. JICSLP '96*).

The foundations of a semantics for Agent Systems, based on graph rewriting, are investigated by D. Janssens (Antwerp) and P. Knirsch (Bremen). It has been shown, at least on the basis of a case study, that several notions relevant to agents, such as reactivity, autonomy and pro-active behaviour, can be expressed by features that are natural on the level of a graph rewriting system: termination conditions, priorities etc. These preliminary results point to the possibility to develop a GTS-based High-level language for agents.

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. 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*.

Besides having a high-quality compiler and programming environment, the success of a programming language is highly dependent on the availability of supporting libraries. 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.*

4.3.6 Specification of Software Engineering Models

View concepts The idea of a combined reference model- and view-based specification approach has been proposed recently in the software engineering community. In [EHTE97] and [EEHT97], a specification technique based on graph transformations is presented which supports such a development approach. The use of graphs and graph transformations supports an intuitive understanding and an integration of static and dynamic aspects on a well-defined semantical base. We define a construction for automatic view integration which assumes that the dependencies between different views are described by a reference model. The views and the reference model are kept consistent manually, which is the task of a model manager. All concepts and results are illustrated at the well-known example of a banking system.

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. [GMET00] presents an tool environment where the ideas of ViewPoint-orientec software development are realized in a first approach.

Object rules 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 [Tae99a], 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.

Consistency of software models In object-oriented software development, requirements of different stakeholders are often manifested in use case models which complement the static domain model by dynamic and functional requirements. In the course of development, these requirements are analyzed and integrated to produce a consistent overall requirements specification. Iterations of the model may be triggered by conflicts between requirements of different parties. However, due to the diversity, incompleteness, and informal nature, in particular of functional and dynamic requirements, such conflicts are difficult to find. Formal approaches to requirements engineering, often based on logic, attack these problems, but require highly specialized experts to write and reason about such specifications. In [HHT02], a formal interpretation of use case models is proposed consisting of UML use case, activity, and collaboration diagrams. The formalization, which is based on concepts from the theory of graph transformation, allows to make precise the notions of conflict and dependency between functional requirements expressed by different use cases. Then, use case models can be statically analyzed, and conflicts or dependencies detected by the analysis can be communicated to the modeler by annotating the model. An implementation of the static analysis within a graph transformation tool is presented.

The use of UML for software specification leads usually to lots of diagrams showing different aspects and components of the software system in several views. In [Tsi00], consistency analysis between class and sequence diagrams based on attributed typed graphs and their transformation is described. More precisely, class diagrams are translated into an attributed typed graph called class graph and the multiplicity adornments into application constraints called multiplicity constraints. Sequence diagrams are represented by a graph grammar called interaction graph grammar. The consistency checking comprises existence, visibility and multiplicity checking. For consistency analysis consistency checking techniques of the algebraic theory of graph grammars using algebraic specifications for the attribute components are used to develop an algorithm which allows us to analyse and check this kind of consistency.

In order to support a view-oriented approach to system modeling, consistency in views and in between views has to be manageable. It is a reasonable approach to consistency management when first choosing a suitable semantic domain, provide a partial mapping into this domain, and specify as well as verify consistency constraints formulated in that domain. Annotated meta model rules can be used to translate elements of UML models into the semantic domain chosen. In [HKT02], triple graph grammars and attributed graph transformation approaches are considered for the precise definition of meta model rules. Moreover, the tool support for automatic translation is outlined.

Part of the success of the Unified Modeling Language (UML) as a specification language is due to its diagrammatic nature. Its meaning is expressed by its meta model, a combination of class diagrams and constraints written in the Object Constraint Language (OCL), a textual language of expressions. Recent efforts have tried to give a formal semantics to OCL in a classical way. In [BKPT00], a graph-based semantics for OCL and a systematic translation of OCL constraints into expressions over graph rules is proposed. Besides providing a semantical formalization of OCL, this translation can be employed to check the consistency of UML model instances with respect to the constraints, using a general purpose graph transformation machine like AGG or PROGRES.

Integration of specification techniques In a software system development process a variety of heterogeneous viewpoint models of the system are constructed for the abstract specification of its functionality and behaviour. These have to be integrated in order to achieve a coherent and consistent global system specification. Transformation systems constitute a formal semantic domain where specification languages conforming to different paradigms can be interpreted, which makes possible a formal semantic integration of heterogeneous viewpoint models. [GR00] and [GJS02] discuss the transformation system approach. Analogously to graph transformation systems states are described by algebraic structures and behaviour by the ordering of state transformation steps, i.e., changes of the algebraic structures. In contrast with graph transformation systems, however, transformation systems are generic with respect to the choice of the state representations, i.e., instead of graphs or extensions thereof arbitrary structures and corresponding logics may be used. Moreover, development relations and structuring operations are defined, which altogether yields a powerful and flexible integration framework.

Distributed software development and software evolution In [Gru99], 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.

The evolution of software has become a more and more critical point in the use and maintenance of large software system. Accordingly, the continuous change of the system's design gets into focus. In [PEB00] and [EBP01] a two-fold approach combining visual architecture description with graphical specification techniques is suggested. Rule-based modification (a generalization of graph transformations) at both levels can be used to describe the model evolution. Allowing transformations at both levels allows distinguishing evolution steps.

Software configuration management 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. With this approach it is possible not only to precisely capture the semantics of individual policies, but also to describe, for the first time, formal properties of the relationship between policies.

Agent-based systems In [KK00a] it is shown that a simple agent system can be modeled by applying graph transformation. Similar to transformation units, agents have local sets of graph rules determining their capabilities and a local control that coordinates the order the rules are applied in. 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.

Agents and agent systems are a new means for structuring software. Concerning this approach an agent is the main abstraction, an entity encapsulating an inner state, a goal, and a set of rules. A first approach towards describing agents and agent systems using rule-based graph transformation is [KK00a]. The idea of a graph transformation-based specification of changes in environments and of agent systems in particular gave rise to further investigations in the context of logistics management ([EKT00, EK00]).

Strongly connected with the idea of a graph-transformation-based specification of changes in environments and of agent systems in particular, investigations in the context of logistics management have been furthered in the direction of Mass Customisation [TKH+01, TWK+01] with local and a global range.

In [BPS00], 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 importance of embedded systems for our daily life is rapidly increasing. More or less unnoticed they fulfill the task of controlling the "behaviour" of many technical systems. Embedded real-time systems play a special role. One important field of application for embedded real-time systems is in the telecommunications industry. The complexity of these systems is rapidly increasing while at the same time the software part is becoming more and more important. In the E-CARES (Ericsson Communication ARchitecture for Embedded Systems) the subject of study is Ericsson's Mobile-service Switching Center (MSC) in order to develop methods, concepts, and tools to support the processes of understanding and restructuring complex legacy telecommunication systems. The paper [HMJ00, MH01a, MH02] reports on the latest developments on this topic.

In [DJM02] the use of graph rewiting for the formalization of refactoring operations is explored. A graph representation of object-oriented programs is presented, and it is demonstrated that a number of refactoring operations considered in the literature can be modeled in an elegant way by graph productions. The examples are based on JAVA, but the main ideas are language-independent. This type of research is important because much of the work on refactoring is still at the semi-formal level, in spite of the fact that it is very important that refactorings guarantee a number of relevant program properties.

4.3.7 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.

In [AKP97] a survey of confluence properties of (acyclic) term graph rewriting is presented. Results and counterexamples are given for plain applications of rewrite rules, extensions with the operations of collapsing and copying, and with both operations together. Collapsing and copying together constitute bisimilarity of term graphs. These results are extended in [AKP00] by considering term graph rewriting *modulo bisimilarity*. This notion of rewriting turns out to be a proper extension of term rewriting but behaves with respect to termination and confluence exactly like term rewriting.

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

A survey of term graph rewriting is given in [Plu99b], where emphasis is laid on 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 article [Plu02] summarizes essentials of term graph rewriting concerning soundness, completeness, termination and confluence.

In [HP02c], the authors review the concept of term graph narrowing as an approach for solving equations by transformations on term graphs. Term graph narrowing combines term graph rewriting with first-order term unification. This mechanism is complete for all term rewriting systems over which term graph rewriting is normalizing and confluent. This includes, in particular, all convergent term rewriting systems. Completeness means that for every solution of a given equation, term graph narrowing can find an equivalent or more general solution. The general motivation for using term graphs instead of terms is to improve efficiency: sharing common subterms saves space and avoids the repetition of computations.

The papers [CD97, CG98a, CG97, CG99a] address the issue of relating (possibly ciclic) term graph rewriting with suitable extensions of term rewriting. Thus they contribute to the semantic foundation of the implementation of programming languages based on term graph rewriting.

In [CD97] it is shown that the rewriting of cyclic term graphs (which denote rational, possibly infinite terms) is adequate with respect to rational parallel term rewriting. The approach constrasts with the "classical" one in that a single graph reduction corresponds to a single (but possibly infinite) parallel term rewrite step whereas, classically, a single graph reduction may correspond to an infinite sequence of term rewrite steps. A related result is presented in [CG98a], where infinite parallel term rewriting is shown to be an adequate semantics for the rewriting of μ-terms, i.e., terms enriched with an auto-instantiation operator.

An original 2-categorical presentation of term graph rewriting is proposed in [CG97], along the lines of the well known description of term rewriting systems as Cartesian 2-categories, where horizontal arrows represent terms, and cells represent rewriting sequences. This presentation relies on the main result of [CG99a], where it is shown that term graphs over a signature Σ are one-to-one with the arrows of the free gs-monodial category generated by Σ. Such a category satisfies all axioms for Cartesian categories but for the naturality of two transformations providing in this way an abstract and clear relationship between terms and term graphs. This work is extended in [CG98b] to a 2-categorical presentation of possibly cyclic term graph rewriting, showing that this presentation is equivalent to the well-accepted operational definition proposed by Barendregt et. al., but for the case of *circular redexes*, for which a different treatment is proposed (and justify formally).

Term graphs also play a central role in the functorial presentation of multi-algebras. Multi-algebras allow to model nondeterminism in an algebraic framework by interpreting operators as functions from individual arguments to sets of possible results. In [CG99b] and its full version [CG02], 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.

Relations, multirelations, partial functions, equivalences can be seen as particular instances of the more general concept of *span* (over **Set**). A span consists of a support (a set) and two legs (two arrows in **Set** having the support as source). In [BG01] the gs-monoidal structures of several span categories is explored, making the correspondence with other classical categories such as **Rel** precise.

Building on term graphs and other algebraic structures for expressing formalisms different from the standard tree-like presentation of terms, where e.g. name sharing, name creation and name hiding can be straightforwardly modeled, it has been possible to identify a kernel of ingredients for describing normal form presentations of such formalisms (namely, *links*, *interfaces* and *modules*), showing that they all form different flavors of enriched symmetric monoidal categories [BGM99b, BGM02].

4.3.8 Application to Neural Networks

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

In [FKB98b] a topology changing training algorithm for Probabilistic

Neural Networks, more exactly the Dynamic Decay Adjustment (DDA) algorithm, is specified by graph transformation rules. With the help of concepts developed in the field of graph transformations the termination and convergence of the DDA is shown.

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

4.3.9 Application to Molecular Computing

Molecular Computing is an exciting, novel and fast growing research area rooted in computer science, but very interdisciplinary in nature (in particular, the interaction with molecular biology, biochemistry and nanotechnology is very vivid and successful). One of the interesting models of molecular computing are *Membrane Systems* which abstract from the rich membrane structure of living cells. Here, the membranes divide the cell space into separate regions. Each region has its own (computational) process going on, and the (process of the) regions can communicate through the membranes that separate/join them. The graph transformation methods turned out to be very suitable in the investigation of the computational nature of membrane systems. The results we have obtained are reported in [PR02, MPPR02].

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.

[AKP97] Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Confluent rewriting of bisimilar term graphs. In *Proc. 4th Workshop on Expressiveness in Concurrency (EXPRESS'97)*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1997. 18 pages.

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

[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.* PhD thesis, Department of Computer Science - University of Pisa, 2000. TD-1/00.

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

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

[Bar00] R. Bardohl. GenGEd – *Visual Definition of Visual Languages based on Algebraic Graph Transformation.* Verlag Dr. Kovac, 2000. PhD thesis, Technical University of Berlin, Dept. of Computer Science, 1999.

[Bar02] R. Bardohl. A Visual Environment for Visual Languages. *Science of Computer Programming (SCP)*, 2002. To appear.

[BBCP00] P. Baldan, N. Busi, A. Corradini, and G.M. Pinna. Functorial concurrent semantics for Petri nets with read and inhibitor arcs. In C. Palamidessi, editor, *Proc. CONCUR'00*, volume 1877 of *Lecture Notes in Computer Science*, pages 442–457. Springer, 2000.

[BC02a] Michel Bauderon and Andrea Corradini, editors. *Proc. GETGRATS Closing Workshop*, volume 51 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.

[BC02b] P. Bottoni and G. Costagliola. On the definition of visual languages and their editors. In M. Hegarty, B. Meyer, and N.H. Narayanan, editors, *Proc. Diagrammatic Representation* *and Inference (DIAGRAMS 2002)*, volume 2317 of *Lecture Notes in Artificial Intelligence*, pages 305–319. Springer, 2002.

[BCE+99] P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent Semantics of Algebraic Graph Transformation Systems. In Ehrig et al. [EKMR99], pages 107–187.

[BCK01] P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K.G. Larsen and M. Nielsen, editors, *Proc. CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2001.

[BCK02] P. Baldan, A. Corradini, and B. König. Static analysis of distributed systems with mobility specified by graph grammars—a case study. In *Proc. IDPT*, 2002. To appear.

[BCM98a] P. Baldan, A. Corradini, and U. Montanari. Concatenable graph processes: relating processes and derivation traces. In K. Larsen and M. Nielsen, editors, *Proc. ICALP'98*, volume 1443, pages 283–295. Springer, 1998.

[BCM98b] P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Proc. FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 1998.

[BCM98c] P. Baldan, A. Corradini, and U. Montanari. An unfolding semantics for DPO graph transformation systems. In Engels and Rozenberg [ER98].

[BCM99] P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proc. FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 1999.

[BCM00] P. Baldan, A. Corradini, and U. Montanari. History preserving bisimulations for contextual nets. In D. Bert and C. Choppy, editors, *Proc. WADT'99*, volume 1827 of *Lecture Notes in Computer Science*, pages 291–310. Springer, 2000.

[BCM01] P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 1(171):1–49, 2001.

[BCM02] P. Baldan, A. Corradini, and U. Montanari. Bisimulation equivalences for graph grammars. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg [on occasion of his 60th birthday, March 14, 2002]*, volume 2300 of *Lecture Notes in Computer Science*, pages 158–190. Springer, 2002.

[BdFMM00a] R. Bruni, D. de Frutos-Escrig, N. Martí-Oliet, and U. Montanari. Bisimilarity congruences for open terms and term graphs via tile logic. In C. Palamidessi, editor, *Proc. 11th Int. Conference on Concurrency Theory (CONCUR 2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000.

[BdFMM00b] R. Bruni, D. de Frutos-Escrig, N. Martí-Oliet, and U. Montanari. Tile bisimilarity congruences for open terms and term graphs. Technical Report TR-00-06, Computer Science Department, University of Pisa, 2000.

[BE00] R. Bardohl and H. Ehrig. Conceptual Model of the Graphical Editor GenGEd for the Visual Definition of Visual Languages. In Ehrig et al. [EEKR00], pages 252–266.

[BE01] R. Bardohl and C. Ermel. Visual Specification and Parsing of a Statechart Variant using GenGEd. In *Proc. Symposium on Visual Languages and Formal Methods (VLFM'01)*, Stresa, Italy, September 5–7 2001.

[BEE01] R. Bardohl, C. Ermel, and H. Ehrig. Generic Description of Syntax, Behavior and Animation of Visual Models. Technical Report 2001/19, Technische Universität Berlin, 2001. ISSN 1436-9915.

[BEE+02] R. Bardohl, K. Ehrig, C. Ermel, A. Qemali, and I. Weinhold. GenGEd – Specifying Visual Environments based on Visual Languages. In Kreowski and Knirsch [KK02], pages 71–82.

[BEH+02] B. Braatz, H. Ehrig, K. Hoffmann, J. Padberg, and M. Urbášek. Application of Graph Transformation Techniques to the Area of Petri Nets. In Kreowski and Knirsch [KK02], pages 35–43.

[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. A Modular Approch to Animation of Simulation Models. In *Proc. 14^{th} Brazilian Symposium on Software Engineering*, Joao Pessoa, Brazil, October 2000.

[BG01] R. Bruni and F. Gadducci. Some algebraic laws for spans (and their connections with multirelations). In W. Kahl, D.L. Parnas, and G. Schmidt, editors, *Proc. Relational Methods in Software (RelMiS 2001)*, volume 44.3 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2001. Also published as Technical Report, Bericht Nr. 2001-02, Fakultät für Informatik, Universität der Bundeswehr München.

[BGM99a] 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.

[BGM99b] R. Bruni, F. Gadducci, and U. Montanari. Normal forms for partitions and relations. In J.L. Fiadeiro, editor, *Proc. 13th Workshop on Recent Trends in Algebraic Development Techniques (WADT'98)*, volume 1589 of *Lecture Notes in Computer Science*, pages 31–47. Springer, 1999.

[BGM02] R. Bruni, F. Gadducci, and U. Montanari. Normal forms for algebras of connections. *Theoretical Computer Science*, 2002. To appear.

[BGRS01] R. Bardohl, M. Große-Rhode, and M. Simeoni. Interactive Rule-based Specification with an Application to Visual Language Definition. In *Proc. Int. Workshop on Algebraic Development Techniques (WADT'01)*, pages 3–4, April 2001.

[BH01] Giorgio Busatto and Berthold Hoffmann. Comparing notions of hierarchical graph transformation. In Baresi et al. [BPT01]. 8 pages.

[BHLM01] R. Bruni, F. Honsell, M. Lenisa, and M. Miculan. Comparing higher-order encodings in logical frameworks and tile logic. In U. Montanari, editor, *Proc. Final Workshop of the TOSCA Project (TOSCA 2001)*, volume 62 of *Electronic Notes in Theoretical Computer Science*, 2001.

[BJK02] Michel Bauderon, Hélène Jacquet, and Renate Klempien-Hinrichs. Pullback rewriting and applications. In Bauderon and Corradini [BC02a]. 10 pages.

[BJSW01] Simon Becker, Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel. A delegation-based model for distributed software process management. In Vincenzo Ambriola, editor, *Proc. 8th European Workshop on Software Process Technology (EWSPT 2001)*, volume 2077 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2001.

[BK98] R. Bardohl and L. Korff-Ribeiro. Specifying a Graph Grammer Editor for Reactive Systems. In *Proc. Workshop Brasileiro de Metodos Formais*, Porto Alegre, Brazil, October 1998.

[BKK01] Giorgio Busatto, Hans-Jörg Kreowski, and Sabine Kuske. Abstract hierarchical graph transformation. Technical Report 1/01, Universität Bremen, 2001.

[BKPT00] P. Bottoni, M. Koch, F. Parisi-Presicce, and G. Taentzer. Consistency Checking and Visualization of OCL Constraints. In A. Evans and S. Kent, editors, *UML 2000 - The Unified Modeling Language*, volume 1939 of *Lecture Notes in Computer Science*. Springer, 2000.

[BKPT01] P. Bottoni, M. Koch, F. Parisi-Presicce, and G. Taentzer. A Visualization of OCL using Collaborations. In M. Gogolla and C. Kobryn, editors, *UML 2001 – The Unified Modeling Language*, volume 2185 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2001.

[BKPT02] P. Bottoni, M. Koch, F. Parisi-Presicce, and G. Taentzer. Working on OCL with Graph Transformation. In Kreowski and Knirsch [KK02], pages 1–10.

[BM97] R. Bruni and U. Montanari. Zero-safe nets, or transition synchronization made simple. In C. Palamidessi and J. Parrow, editors, *Proc. 4th Workshop on Expressiveness in Concurrency (EXPRESS'97)*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1997.

[BM98] R. Bruni and U. Montanari. Zero-safe nets: The individual token approach. In F. Parisi-Presicce, editor, *Proc. 12th Workshop on Recent Trends in Algebraic Development Techniques (WADT'97)*, volume 1376 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 1998.

[BM99a] R. Bruni and U. Montanari. Cartesian closed double categories, their lambda-notation, and the pi-calculus. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 246–265. IEEE Computer Society Press, 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 M. Nielsen and D. Simpson, editors, *Proc. 21st Int. Conf. on Application and Theory of Petri Nets (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2000.

[BM00b] R. Bruni and U. Montanari. First order and higher order tile models for open and mobile systems (abstract). In *Proc. Workshop Annuale del Progetto TOSCA (TOSCA'00)*, 2000. Virtual Proceedings.

[BM00c] R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Information and Computation*, 156(1-2):46–89, 2000.

[BM00d] R. Bruni and U. Montanari. Zero-safe nets: Modeling transactions via transition synchronization (abstract). Technical Report CS-2000-01, Computer Science Department, University Ca' Foscari of Venice, 2000.

[BM01a] R. Bruni and U. Montanari. Transactions and zero-safe nets. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, volume 2128 of *Lecture Notes in Computer Science*, pages 380–426. Springer, 2001.

[BM01b] R. Bruni and U. Montanari. Zero-safe net models for transactions in Linda. In U. Montanari and V. Sassone, editors, *Proc. Int. Workshop on Concurrency and Coordination (ConCoord 2001)*, volume 54 of *Electronic Notes in Theoretical Computer Science*, 2001.

[BM02] R. Bruni and U. Montanari. Dynamic connectors for concurrency. *Theoretical Computer Science*, 281(1-2):131–176, 2002.

[BMM98a] R. Bruni, J. Meseguer, and U. Montanari. Implementing tile systems: Some examples from process calculi. In P. Degano, U. Vaccaro, and G. Pirillo, editors, *Proc. 6th Italian Conference on Theoretical Computer Science (ICTCS'98)*, pages 168–179. World Scientific, 1998.

[BMM98b] R. Bruni, J. Meseguer, and U. Montanari. Internal strategies in a rewriting implementation of tile systems. In C. Kirchner and H. Kirchner, editors, *Proc. 2nd Workshop on Rewriting Logic and its Applications (WRLA'98)*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1998.

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

[BMM99] R. Bruni, J. Meseguer, and U. Montanari. Executable tile specifications for process calculi. In J.-P. Finance, editor, *Proc. Fundamental Approaches to Software Engineering (FASE'99)*, volume 1577 of *Lecture Notes in Computer Science*, pages 60–76. Springer, 1999.

[BMM02] R. Bruni, J. Meseguer, and U. Montanari. Symmetric monoidal and cartesian double categories as a semantic framework for tile logic. *Mathematical Structures in Computer Science*, 12(1):53–90, 2002.

[BMMS98] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. A comparison of Petri net semantics under the collective token philosophy. In J. Hsiang and A. Ohori, editors, *Proc. 4th Asian Computing Science Conference (ASIAN'98)*, volume 1538 of *Lecture Notes in Computer Science*, pages 225–244. Springer, 1998.

[BMMS99] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for Petri nets under the individual token philosophy. In M. Hofmann, G. Rosolini, and D. Pavlovic, editors, *Proc. 8th Category Theory and Computer Science (CTCS'99)*, volume 29 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1999.

[BMMS01] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial models for Petri nets. *Information and Computation*, 170(2):207–236, 2001.

[BMR01] R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Theory and Practice of Logic Programming*, 1(6):647–690, 2001.

[BMS00] R. Bruni, U. Montanari, and V. Sassone. Open ended systems, dynamic bisimulation and tile logic. In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *Proc. IFIP Int. Conf. on Theoretical Computer Science (IFIP TCS 2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2000.

[BNS00] R. Bardohl, M. Niemann, and M. Schwarze. GenGEd – A Development Environment for Visual Languages. In Nagl et al. [NSM00], pages 233–240.

[BPS00] P. Bottoni, F. Parisi-Presicce, and M. Simeoni. From formulae to rewriting systems. In Ehrig et al. [EEKR00], pages 267–280.

[BPT01] Luciano Baresi, Mauro Pezzè, and Gabriele Taentzer, editors. *Proc. ICALP 2001 Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2001)*, volume 50.3 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, July 2001.

[BR01] R. Bardohl and L. Ribeiro. Towards a Compositional Approach to Define Graphical Animation of Software Applications. In *Proc. Symposium on Visual Languages and Formal Methods (VLFM'01)*, Stresa, Italy, September 5–7 2001.

[Bru99] R. Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems*. PhD thesis, Computer Science Department, University of Pisa, 1999.

[BS00] R. Bruni and V. Sassone. Algebraic models for contextual nets. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Proc. 27th Int. Coll. on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2000.

[BS01] R. Bruni and V. Sassone. Two algebraic process semantics for contextual nets. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, volume 2128 of *Lecture Notes in Computer Science*, pages 427–456. Springer, 2001.

[BST00a] P. Bottoni, A. Schürr, and G. Taentzer. Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. In *Proc. IEEE Symposium on Visual Languages*, September 2000.

[BST00b] P. Bottoni, A. Schürr, and G. Taentzer. Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. Si-2000-06, University of Rome, 2000.

[BST01] R. Bardohl, T. Schultzke, and G. Taentzer. Visual Language Parsing in GenGEd. In Baresi et al. [BPT01].

[BT97] R. Bardohl and G. Taentzer. Defining Visual Languages by Algebraic Specification Techniques and Graph Grammars. In *Proc. Theory of Visual Languages '97*, pages 27–42, 1997.

[BT01] P. Bottoni and G. Taentzer. Parsing and Semantics of a Statechart Variant by Contextual Layered Graph Transformation. In *Proc. Symposium on Visual Languages and Formal Methods (Statechart Modeling Contest), IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01)*, Stresa, Italy, 2001.

[BtH00] Giorgio Busatto and Pieter Jan 't Hoen. A graph-grammar based approach for the specification of hypermedia application dynamics. In Corradini and Heckel [CH00], pages 403–409.

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

[Bus02a] Giorgio Busatto. *An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation*. PhD thesis, University of Paderborn, 2002.

[Bus02b] Giorgio Busatto. Modeling hyperweb dynamics through hierarchical graph transformation. In Kreowski and Knirsch [KK02], pages 141–150.

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

[CG97] A. Corradini and F. Gadducci. A 2-categorical presentation of term graph rewriting. In *Proc. CTCS'97*, volume 1290 of *Lecture Notes in Computer Science*. Springer, 1997.

[CG98a] A. Corradini and F. Gadducci. Rational term rewriting. In M. Nivat, editor, *Proc. FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 1998.

[CG98b] A. Corradini and F. Gadducci. Rewriting on cyclic structures. Technical Report TR-98-05, Dipartimento di Informatica, Pisa, 1998. Extended abstract in Proc. FICS'98.

[CG99a] A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7:299–331, 1999.

[CG99b] A. Corradini and F. Gadducci. Functorial Semantics of Multialgebras. In J.L. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques*, volume 1589 of *Lecture Notes in Computer Science*. Springer, 1999.

[CG02] A. Corradini and F. Gadducci. A functorial semantics for multi-algebras and partial algebras, with applications to syntax. *Theoretical Computer Science*, 2002. 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.

[CH00] A. Corradini and R. Heckel, editors. *Proc. ICALP 2000 Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2000)*, Geneva, Switzerland, July 2000. Carleton Scientific.

[CHM00] A. Corradini, R. Heckel, and U. Montanari. Graphical operational semantics. In Corradini and Heckel [CH00]. `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/CorradiniGTVMT00.pdf`.

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

[Cre99] K. Cremer. *Graphbasierte Werkzeuge zum Reverse Engineering und Reengineering.* Dissertation, RWTH Aachen, 1999.

[Cre00] Katja Cremer. Graph-based reverse engineering and reengineering tools. In Nagl et al. [NSM00], pages 95–110.

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

[DEKK00a] Frank Drewes, Sigrid Ewert, Renate Klempien-Hinrichs, and Hans-Jörg Kreowski. Computing raster images from grid picture grammars. Technical Report 2/00, Fachbereich Mathematik/Informatik, Universität Bremen, 2000.

[DEKK00b] Frank Drewes, Sigrid Ewert, Renate Klempien-Hinrichs, and Hans-Jörg Kreowski. Computing raster images from grid picture grammars. In M. Daley, M.G. Eramian, and S. Yu, editors, *Preproc. 5th Int. Conference on Implementation and Application of Automata (CIAA 2000)*, pages 79–85. University of Western Ontario, London, Canada, 2000.

[DEKK01] Frank Drewes, Sigrid Ewert, Renate Klempien-Hinrichs, and Hans-Jörg Kreowski. Computing raster images from grid picture grammars. In S. Yu and A. Păun, editors, *Proc. 5th Int. Conference on Implementation and Application of Automata (CIAA 2000)*, volume 2088 of *Lecture Notes in Computer Science*, pages 113–121. Springer, 2001.

[DH00a] R. Depke and R. Heckel. Formalizing the development of agent-based systems using graph processes. In Corradini and Heckel [CH00], pages 419–426. http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/DepkeGTVMT00.pdf.

[DH00b] R. Depke and R. Heckel. Towards role-based modeling of autonomous agents. In H. Giese and S. Philippi, editors, *Visuelle Verhaltensmodellierung verteilter und nebenläufiger Software-Systeme, Workshop des Arbeitskreises GROOM der GI Fachgruppe 2.1.9 Objektorientierte Software-Entwicklung*. Universität Münster, 2000.

[DHK00a] R. Depke, R. Heckel, and J. M. Küster. Integrating visual modeling of agent-based and object-oriented systems. In *Proc. Fourth Int. Conference on Autonomous Agents (AGENTS-2000)*, pages 82–83, Barcelona, Spain, June 2000. http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/DepkeAgents00.pdf.

[DHK00b] R. Depke, R. Heckel, and J.M. Küster. Requirement specification and design of agent-based systems with graph transformation, roles and UML. In P. Ciancarini and M.J. Wooldridge, editors, *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (ICSE-2000 Workshop)*, Limerick, Ireland, June 2000. University of Limerick. http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/DepkeAOSE00.pdf.

[DHK01a] R. Depke, R. Heckel, and J. M. Küster. Improving the agent-oriented modeling process with roles. In *Proc. 5th Int. Conference on Autonomous Agents (AGENTS-2001)*, Proc. of the ACM, pages 640–647, Montreal, Canada, June 2001. ACM.

[DHK01b] R. Depke, R. Heckel, and J. M. Küster. Roles in agent-oriented modeling. *Int. Journal of Software Engineering and Knowledge Engineering*, 11(3):281–302, 2001.

[DHK01c] R. Depke, R. Heckel, and J.M. Küster. Agent-oriented modeling with graph transformation. In P. Ciancarini and M.J. Wooldridge, editors, *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000), Limerick, Ireland, June 2000*, volume 1957 of *LNCS*, pages 105–120. Springer, 2001. http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/DepkeAOSE00.pdf.

[DHM02] Frank Drewes, Berthold Hoffmann, and Mark Minas. Constructing shapely nested graph transformations. In Kreowski and Knirsch [KK02], pages 107–118.

[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, 2000.

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

[DHP02] Frank Drewes, Berthold Hoffmann, and Detlef Plump. Hierarchical graph transformation. *Journal of Computer and System Sciences*, 64(2):249–283, 2002.

[DJM02] S. Demeyer, D. Janssens, and T. Mens. Formalizing behaviour preserving program transformations. Technical Report 02–03, University of Antwerp - UIA, dept. of Math. and Comp. Sci., 2002.

[DK99] Frank Drewes and Hans-Jörg Kreowski. Picture generation by collage grammars. In Ehrig et al. [EEKR99], chapter 11, pages 397–458.

[DK00a] Frank Drewes and Renate Klempien-Hinrichs. Picking knots from trees. The syntactic structure of celtic knotwork. Technical Report 4/00, Universität Bremen, 2000.

[DK00b] Frank Drewes and Renate Klempien-Hinrichs. Picking knots from trees. The syntactic structure of celtic knotwork. In M. Anderson, P. Cheng, and V. Haarslev, editors, *Proc. 1st Int. Conference on Theory and Application of Diagrams (DIAGRAMS 2000)*, volume 1889 of *Lecture Notes in Artificial Intelligence*, pages 89–104. Springer, 2000.

[DK00c] Frank Drewes and Peter Knirsch. TREEBAG – a short presentation. In Nagl et al. [NSM00], pages 411–417.

[DK01a] Frank Drewes and Renate Klempien-Hinrichs. TREEBAG. In S. Yu and A. Păun, editors, *Proc. 5th Int. Conference on Implementation and Application of Automata (CIAA 2000)*, volume 2088 of *Lecture Notes in Computer Science*, pages 329–330. Springer, 2001.

[DK01b] Frank Drewes and Hans-Jörg Kreowski. Reading words in graphs generated by hyperedge replacement. In C. Martin-Vide and V. Mitrana, editors, *Where Mathematics, Computer Science, Linguistics and Biology Meet*, chapter 22, pages 243–252. Kluwer Academic Publishers, 2001.

[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)*, pages 326–337. World Scientific, 2000.

[DKKK00] Frank Drewes, Peter Knirsch, Hans-Jörg Kreowski, and Sabine Kuske. Graph transformation modules and their composition. In Nagl et al. [NSM00], pages 15–30.

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

[Dre97] Frank Drewes. On the generation of trees by hyperedge replacement. In I. Prívara and P. Ružička, editors, *Proc. Mathematical Foundations of Computer Science (MFCS'97)*, volume 1295 of *Lecture Notes in Computer Science*, pages 229–238. Springer, 1997.

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

[Dre98b] Frank Drewes. Some remarks on the generative power of collage grammars and chain-code grammars. In Engels and Rozenberg [ER98], pages 425–432.

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

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

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

[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 Ehrig et al. [EEKR00], pages 1–14.

[Dre00b] Frank Drewes. Tree-based picture generation. *Theoretical Computer Science*, 246:1–51, 2000.

[Dre01] Frank Drewes. Tree-based generation of languages of fractals. *Theoretical Computer Science*, 262:377–414, 2001.

[EBE01] C. Ermel, R. Bardohl, and H. Ehrig. Specification and Implementation of Animation Views for Petri Nets. In Weber et al. [WER01], pages 75–92.

[EBP01] C. Ermel, R. Bardohl, and J. Padberg. Visual Design of Software Architecture and Evolution based on Graph Transformation. In Ehrig et al. [EEP01].

[EEHT97] G. Engels, H. Ehrig, R. Heckel, and G. Taentzer. A combined reference model- and view-based approach to system specification. *Int. Journal of Software and Knowledge Engineering*, 7(4):457–477, 1997.

[EEHT98] H. Ehrig, G. Engels, R. Heckel, and G. Taentzer. Classification and comparison of modularity concepts for graph transformation systems. In Engels and Rozenberg [ER98], pages 122–131.

[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, 2000.

[EEP01] Hartmut Ehrig, Claudia Ermel, and Julia Padberg, editors. *Proc. Uniform Approaches to Graphical Process Specification Techniques (UNIGRA 2001), Satellite Workshop to ETAPS 2001*, volume 44.4 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2001.

[EH00a] G. Engels and R. Heckel. From trees to graphs: Defining the semantics of diagram languages with graph transformation. In Corradini and Heckel [CH00]. `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/EngelsGTVMT00.pdf`.

[EH00b] G. Engels and R. Heckel. Graph transformation as unifying formal framework for system modeling and model evolution. In *Proc. ICALP 2000*, Lecture Notes in Computer Science, Geneva, Switzerland, July 2000. Springer. `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/EngelsIcalp00.pdf`.

[EHHS00] G. Engels, J.H. Hausmann, R. Heckel, and St. Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In A. Evans, S. Kent, and B. Selic, editors, *Proc. UML 2000*, volume 1939 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2000.

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach. In Rozenberg [Roz97], pages 247–312.

[EHTE97] G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A view-oriented approach to system modelling using graph transformation. In *Proc. ESEC/FSE'97, Zürich*, volume 1301 of *Lecture Notes in Computer Science*, pages 327–343. Springer, 1997.

[EK00] Jens Eschenbächer and Peter Knirsch. Reengineering of supply chain networks with change management. In *Global Logistics for the New Millennium – Proc. 5th Int. Symposium on Logistics (ISL 2000), Iwate, Japan*, pages 198–207, 2000.

[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.

[EKT00] Jens Eschenbächer, Peter Knirsch, and Ingo J. Timm. Demand chain optimisation by using agent technology. In *IFIP WG 5.7 – Int. Conference on Integrated Production Management: Information and Communication Technology in Logistics and Production Management, Tromsø, Norway*, pages 285–292, 2000.

[EO01] H. Ehrig and F. Orejas. A generic component concept for integrated data type and process specification techniques. Technical Report 2001/12, Technische Universität Berlin, FB Informatik, 2001.

[EOB+02a] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A generic component concept for system modelling. In R. Kutsche and H. Weber, editors, *Proc. Formal Aspects of Software Engineering (FASE 2002)*, 2002.

[EOB+02b] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A transformation-based component framework for a generic integrated modeling technique. Technical Report 2002/2, ISSN 1436-9915, Technische Universität Berlin, FB Informatik, 2002.

[ER98] Gregor Engels and Gregorz Rozenberg. *Preproc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT '98).* Technical Report tr-ri-98-201, Computer Science and Mathematics Department, University of Paderborn, Germany, 1998.

[ERT99] C. Ermel, M. Rudolf, and G. Taentzer. The AGG-Approach: Language and Tool Environment. In Ehrig et al. [EEKR99], pages 551–603.

[ET00a] H. Ehrig and G. Taentzer. A Proposal for Consistent Integration of Visual Computing with Visual Development of Software Components. In *Proc. Satellite Workshops of 27th Int. Colloqium on Automata, Languages, and Programming (ICALP 2000)*, pages 427–434, Geneva, Switzerland, July 2000. Carleton Scientific, Canada.

[ET00b] H. Ehrig and G. Taentzer. *Proc. Joint APPLIGRAPH/ GETGRATS Workshop on Graph Transformation (GRATRA 2000).* Technical Report 2000-2, Comp. Sci. Dept., Technical University of Berlin, Germany, March 2000.

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

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

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

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

[FKT00] I. Fischer, M. Koch, and G Taentzer. Local Views on Distributed Systems and Their Communications. In Ehrig et al. [EEKR00], pages 164–178.

[FKTV99] I. Fischer, M. Koch, G. Taentzer, and V. Volle. Distributed Graph Transformation with Application to Visual Design of Distributed Systems. In Ehrig et al. [EKMR99], pages 269–340.

[FM97a] G.L. Ferrari and U. Montanari. A tile-based coordination view of asynchronous pi-calculus. In I. Prívara and P. Ruzicka, editors, *Proc. 22nd Int. Symposium on Mathematical Foundations of Computer Science (MFCS'97)*, volume 1295 of *Lecture Notes in Computer Science*, pages 52–70. Springer, 1997.

[FM97b] G.L. Ferrari and U. Montanari. Tiles for concurrent and located calculi. In C. Palamidessi and J. Parrow, editors, *Proc. 4th Workshop on Expressiveness in Concurrency (EXPRESS'97)*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1997.

[FM00] G.L. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Information and Computation*, 156(1-2):173–235, 2000.

[FMT01] G. Ferrari, U. Montanari, and E. Tuosto. A LTS Semantics of Ambients via Graph Synchronization with Mobility. In *Proc. ICTCS 2001*, volume 2202 of *Lecture Notes in Computer Science*. Springer, 2001.

[FNTZ00] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A new Graph Grammar Language based on the Unified Modelling Language and Java. In Ehrig et al. [EEKR00].

[Gat00] Felix Gatzemeier. Patterns, schemata, and types — author support through formalized experience. In Bernhard Ganter and Guy W. Mineau, editors, *Proc. Int. Conference on Conceptual Structures 2000*, volume 1867 of *Lecture Notes in Artificial Intelligence*, pages 27–40. Springer, 2000.

[GH02] S. Gyapay and R. Heckel. Towards graph transformation with time. In Kreowski and Knirsch [KK02], pages 131–140.

[GJS02] M. Große-Rhode, S. John, and G. Schröter. Transformation Systems for the Integration of Software Specification. In Kreowski and Knirsch [KK02], pages 151–160.

[GM00a] 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. Also Technical Report TR-27/96, Dipartimento di Informatica, Università di Pisa, 1996.

[GM00b] Felix Gatzemeier and Oliver Meyer. Making complex document structures accessible through templates. In Susan B. Jones and Beth Weise Moeller, editors, *Proc. 18th Annual Conference on Computer Documentation (ipcc/sigdoc 2000)*, pages 509–519, Cambridge, MA, 9 2000. IEEE Professional Communication Society and ACM Special Interest Group on Documentation, IEEE Press.

[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, volume 1785 of *Lecture Notes in Computer Science*. 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 Int. Symposium on Requirements Engineering (RE'99), June 7-11, 1999, University of Limerick, Ireland*. IEEE Computer Society, 1999. ISBN 0-7695-0188-5.

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

[GPS98] M. Große-Rhode, F. Parisi Presicce, and M. Simeoni. Refinements of graph transformation systems via rule expressions. In Engels and Rozenberg [ER98], pages 96–103.

[GPST00] 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 Nagl et al. [NSM00], pages 31–45.

[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.

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

[Gru99] S. Gruner. *Eine schematische und grammatische Korrespondenzmethode zur Spezifikation konsistent verteilter Datenmodelle*. PhD thesis, RWTH Aachen, 1999. Published by Shaker-Verlag Aachen/Maastricht.

[Gru00] S. Gruner. Meta typing is compatible to the typed SPO approach. In Ehrig and Taentzer [ET00b], pages 50–58.

[GSP98a] M. Große-Rhode, M. Simeoni, and F. Parisi Presicce. Refinements and modules for typed graph transformation systems. In *Proc. Workshop on Algebraic Development Techniques (WADT'98), ETAPS'98 Satellite Workshop*, 1998.

[GSP98b] M. Große-Rhode, M. Simeoni, and F. Parisi Presicce. Spatial and temporal refinement of typed graph transformation systems. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proc. Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 553 – 561. Springer, 1998.

[HE00] R. Heckel and G. Engels. Graph transformation and visual modeling techniques. *Bulletin of the European Association of Theoretical Computer Science (EATCS)*, 71:186–202, 2000. `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/HeckelEATCS00.pdf`.

[HE01] R. Heckel and G. Engels. Graph transformation as a meta language for dynamic modeling and model evolution. In T. Mens and M. Wermelinger, editors, *Int. Special Session on Formal Foundations of Software Evolution, Lisboa, Portugal*, March 2001. Co-located with the European Conference on Software Maintainance and Reengineering (CSMR 2001).

[HH01] J.H. Hausmann and R. Heckel. Use cases as views: A formal approach to requirements engineering in the united process. In *Proc. Integrating Diagrammatic and Formal Specification Techniques, Tagungsband der GI/OCG-Jahrestagung*, pages 595–599, Wien, 2001. ISBN 3-85403-157-2.

[HHKK00] Reiko Heckel, Berthold Hoffmann, Peter Knirsch, and Sabine Kuske. Simple modules for GRACE. In Ehrig et al. [EEKR00], pages 383–395.

[HHS01] J.H. Hausmann, R. Heckel, and S. Sauer. Towards dynamic meta modeling of UML extensions: An extensible semantics for UML sequence diagrams. In M. Minas and A. Schürr, editors, *Symposium on Visual Languages and Formal Methods, IEEE Symposia on Human Computer Interaction (HCI 2001), Stresa, Italy*, Los Alamitos, CA, September 2001. IEEE Computer Society Press.

[HHT02] J.H. Hausmann, R. Heckel, and G. Taentzer. Detection of conflicting functional requirements in a use case-driven approach. In *Proc. Int. Conference on Software Engineering (ICSE 2002)*, Orlando, FL, USA, 2002. To appear.

[HIM00] D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of software architecture styles with name mobility. In A. Porto and G.-C. Roman, editors, *Coordination 2000*, volume 1906 of *LNCS*, pages 148–163, 2000.

[HKT02] R. Heckel, J. Küster, and G. Taentzer. Towards automatic translation of UML models into semantic domains. In Kreowski and Knirsch [KK02], pages 11–22.

[HKW97] Peter Heimann, Carl-Arndt Krapp, and Bernhard Westfechtel. An environment for managing software development processes. In *Proc. 8th Conference on Software Engineering Environments*, pages 101–109, Cottbus, Germany, 1997. IEEE Computer Society Press.

[HKWJ97] Peter Heimann, Carl-Arndt Krapp, Bernhard Westfechtel, and Gregor Joeris. Graph-based software process management. *International Journal of Software Engineering and Knowledge Engineering*, 7(4):431–455, December 1997.

[HLEO00] R. Heckel, M. Llabres, H. Ehrig, and F. Orejas. Concurrency of double-pullback graph transitions. In Ehrig and Taentzer [ET00b], pages 146–154.

[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.

[HM00a] D. Hirsch and U. Montanari. Higher-order hyperedge replacement systems and their transformations. In Ehrig and Taentzer [ET00b].

[HM00b] Berthold Hoffmann and Mark Minas. A generic model for diagram syntax and semantics. In Corradini and Heckel [CH00], pages 443–450.

[HM00c] Berthold Hoffmann and Mark Minas. Towards generic rule-based visual programming. In Margaret M. Burnett et al., editor, *Proc. Int. Conference on Visual Languages (VL 2000)*. IEEE Press, 2000.

[HM00d] Berthold Hoffmann and Mark Minas. Towards rule-based visual programming of generic visual systems. In Nachum Dershowitz and Claude Kirchner, editors, *Proc. Workshop on Rule-Based Languages*, 2000.

[HM00e] Berthold Hoffmann and Mark Minas. Über generisches Visuelles Programmieren. In Stephan Diehl and Andreas Kerren, editors, *Proc. GI-Workshop Software Visualization*, Technical Report A/01/2000, pages 41–52. Universität des Saarlandes, 2000. In German.

[HM01a] Dominikus Herzberg and André Marburger. Understanding complex legacy telecommunication systems. In *Proc. 5th European Conference on Software Maintenance and Reengineering (CSMR 2001)*. IEEE Computer Society Press, 2001. ISBN 0-7695-1028-0.

[HM01b] D. Hirsch and U. Montanari. A graphical calculus for name mobility. In *Proc. Workshop on Software Engineering and Mobility*, 2001. Co-located with ICSE 2001, Toronto.

[HM01c] Berthold Hoffmann and Mark Minas. Transformation of shaped nested graphs and diagrams. In M. van den Brand and R. Verma, editors, *Proc. 2nd Int. Workshop on Rule-Based Programming (RULE 2001)*, volume 59(4) of *Electronic Notes in Theoretical Computer Science*, 2001. 15 pages.

[HMJ00] Dominikus Herzberg, André Marburger, and Tony Jokikyyny. E-cares research project: Understanding complex legacy telecommunication systems. In Jürgen Ebert, Bernt Kullbach, and Franz Lehner, editors, *2. Workshop on Software Reengineering*, Fachberichte der Universität Koblenz-Landau, 2000.

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

[HMP00] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout approach with injective matching. In Ehrig et al. [EEKR00], pages 103–116.

[HMP01] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.

[Hof00] Berthold Hoffmann. From graph transformation to rule-based programming with diagrams. In Nagl et al. [NSM00], pages 165–180.

[Hof01] Berthold Hoffmann. Shapely hierarchical graph transformation. In *Proc. IEEE Symposia on Human-Centric Computing Languages and Environments*, pages 30–37. IEEE Computer Press, 2001.

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

[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, 1999.

[HP01] Annegret Habel and Detlef Plump. Computational completeness of programming languages based on graph transformation. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, volume 2030 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2001.

[HP02a] Annegret Habel and Detlef Plump. A core language for graph transformation. In Kreowski and Knirsch [KK02], pages 187–199.

[HP02b] Annegret Habel and Detlef Plump. Relabelling in graph transformation. In *Proc. Int. Conference on Graph Transformation (ICGT 2002)*, Lecture Notes in Computer Science. Springer, 2002. To appear.

[HP02c] Annegret Habel and Detlef Plump. Solving equations by graph transformation. In Bauderon and Corradini [BC02a].

[HS00] R. Heckel and S. Sauer. Dynamische Metamodellierung als Methode zur Definition einer operationalen Semantik für die UML. *Softwaretechnik-Trends*, 20(2):43–44, May 2000.

[HS01] R. Heckel and St. Sauer. Strengthening UML collaboration diagrams by state transformations. In H. Hußmann, editor, *Proc. Fundamental Approaches to Software Engineering (FASE 2001), Genova, Italy*, volume 2185 of *Lecture Notes in Computer Science*. Springer, april 2001.

[HZ01] R. Heckel and A. Zündorf. How to specify a graph transformation approach: A meta model for FUJABA. In Ehrig et al. [EEP01].

[Jäg00] Dirk Jäger. Upgrade - a framework for graph-based visual applications. In Nagl et al. [NSM00], pages 427–432.

[Jan99] D. Janssens. Actor grammars and local actions. In Ehrig et al. [EKMR99], pages 57–106.

[JK00] Hélène Jacquet and Renate Klempien-Hinrichs. Node replacement in hypergraphs: Translating NCE rewriting into the pullback approach. In Ehrig et al. [EEKR00], pages 117–130.

[JSW99] Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel. Using UML for software process modeling. In Oscar Nierstrasz and Michel Lemoine, editors, *Software Engineering — ESEC/FSE '99*, volume 1687 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 1999.

[JSW00] Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel. Ahead: A graph-based system for modeling and managing development processes. In Nagl et al. [NSM00], pages 325–339.

[JV00] D. Janssens and N. Verlinden. A framework for ESM and NLC: Local action systems. In Ehrig et al. [EEKR00], pages 194–214.

[JV01] D. Janssens and N. Verlinden. Modeling Petri nets by local action systems. In Baresi et al. [BPT01].

[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 *Proc. 9th Int. Workshop on Software Specification and Design*, pages 77–86. IEEE Computer Society, 1998.

[KBK01] Hans-Jörg Kreowski, Giorgio Busatto, and Sabine Kuske. GRACE as a unifying approach to graph-transformation-based specification. In Ehrig et al. [EEP01]. 15 pages.

[KBK+02] Hans-Jörg Kreowski, Giorgio Busatto, Renate Klempien-Hinrichs, Peter Knirsch, and Sabine Kuske. Structured modeling with GRACE. In Bauderon and Corradini [BC02a]. 13 pages.

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

[KK99a] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units and modules. In Ehrig et al. [EEKR99], chapter 15, pages 607–638.

[KK99b] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units with interleaving semantics. *Formal Aspects of Computing*, 11(6):690–723, 1999.

[KK00a] Peter Knirsch and Hans-Jörg Kreowski. A note on modeling agent systems by graph transformation. In Nagl et al. [NSM00], pages 79–86.

[KK00b] Hans-Jörg Kreowski and Sabine Kuske. Note on approach-independent structuring concepts for rule-based systems. In Ehrig and Taentzer [ET00b], pages 41–49.

[KK00c] Hans-Jörg Kreowski and Sabine Kuske. Suggestions on the modularization of rule-based systems. In Martin Wirsing, Martin Gogolla, Hans-Jörg Kreowski, Tobias Nipkow, and Wolfgang Reif, editors, *Proc. Rigorose Entwicklung software-intensiver Systeme*, Technical Report Nr. 0005, University of Munich, pages 73–82, 2000.

[KK02] Hans-Jörg Kreowski and Peter Knirsch, editors. *Proc. APPLIGRAPH Workshop on Applied Graph Transformation (AGT 2002)*, ETAPS 2002 Satellite Events. University of Grenoble, 2002.

[KKK02] Renate Klempien-Hinrichs, Peter Knirsch, and Sabine Kuske. Modeling the pickup-and-delivery problem with structured graph transformation. In Kreowski and Knirsch [KK02], pages 119–130.

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

[KKSW00] Carl-Arndt Krapp, Sven Krüppel, Ansgar Schleicher, and Bernhard Westfechtel. Graph-based models for managing development processes, resources, and products. In Ehrig et al. [EEKR00], pages 455–474.

[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 Int. Symposium on Fundamentals of Computation Theory (FCT'99)*, volume 1684 of *Lecture Notes in Computer Science*. Springer, 1999.

[Kle98] Renate Klempien-Hinrichs. Net refinement by pullback rewriting. In M. Nivat, editor, *Proc. Foundations of Software Science and Computation Structures (FoSSaCS'98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 1998.

[Kle99] Renate Klempien-Hinrichs. The generative power of context-free node rewriting in hypergraphs. *Grammars*, 2:211–221, 1999.

[Kle00] Renate Klempien-Hinrichs. Normal forms for context-free node-rewriting hypergraph grammars. In Ehrig and Taentzer [ET00b], pages 224–231.

[Kle01] Renate Klempien-Hinrichs. *Context-free Hypergraph Grammars. Node and Hyperedge Rewriting with an Application to Petri Nets.* Books on Demand, Norderstedt, 2001. Doctoral dissertation, Universität Bremen, 2000.

[Kle02a] Renate Klempien-Hinrichs. Context-free hypergraph grammars with node rewriting. In Bauderon and Corradini [BC02a]. 10 pages.

[Kle02b] Renate Klempien-Hinrichs. Normal forms for context-free node-rewriting hypergraph grammars. *Mathematical Structures in Computer Science*, 12:135–148, 2002.

[KM00] Oliver Köth and Mark Minas. Generating diagram editors providing free-hand editing as well as syntax-directed editing. In Ehrig and Taentzer [ET00b].

[KM01] B. König and U. Montanari. Observational equivalence for synchronized graph rewriting. In N. Kobayashi and B. Pierce, editors, *Proc. TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2001.

[KMP00] M. Koch, L.V. Mancini, and F. Parisi-Presicce. A formal model for role-based access control using graph transformation. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *Proc. 6th European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2000.

[KMP01a] M. Koch, L.V. Mancini, and F. Parisi-Presicce. Foundations for a graph-based approach to the Specification of Access Control Policies. In F. Honsell and M. Miculan, editors, *Proc. Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, volume 2030 of *Lecture Notes in Computer Science*, pages 287–302. Springer, March 2001.

[KMP01b] M. Koch, L.V. Mancini, and F. Parisi-Presicce. On the specification and evolution of access control policies. In S. Osborn, editor, *Proc. 6th ACM Symp. on Access Control Models and Technologies*, pages 121–130. ACM, May 2001.

[KMP02] M. Koch, L.V. Mancini, and F. Parisi-Presicce. Conflict detection and resolution in access control policy specifications. In M. Nielsen and U. Engberg, editors, *Proc. Foundations of Software Science and Computation Structures (FoSSaCS 2002)*, volume 2303 of *Lecture Notes in Computer Science*, pages 223–237. Springer, April 2002.

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

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

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

[Koc99] M. Koch. *Integration of Graph Transformation and Temporal Logic for the Specification of Distributed Systems*. PhD thesis, Technische Universität Berlin, FB 13, 1999.

[Kra98] Carl-Arndt Krapp. *An Adaptable Environment for the Management of Development Processes*. Number 22 in Aachener Beiträge zur Informatik. Augustinus Buchhandlung, Aachen, Germany, 1998.

[Kre02] Hans-Jörg Kreowski. A sight-seeing tour of the computational landscape of graph transformation. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing. Essays Dedicated to Grzegorz Rozenberg*, volume 2300 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2002.

[Kus00a] Sabine Kuske. More about control conditions for transformation units. In Ehrig et al. [EEKR00], pages 323–337.

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

[Kus01] Sabine Kuske. A formal semantics of UML state machines based on structured graph transformation. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 – The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, volume 2185 of *Lecture Notes in Computer Science*, pages 241–256, 2001.

[Kus02] Sabine Kuske. Parameterized transformation units. In Bauderon and Corradini [BC02a]. 12 pages.

[KV00] Hans-Jörg Kreowski and Gabriel Valiente. Redundancy and subsumption in high-level replacement systems. In Ehrig et al. [EEKR00], pages 215–227.

[LB93] M. Löwe and M. Beyer. AGG — an implementation of algebraic graph rewriting. In *Proc. 5th Int. Conf. Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 451–456. Springer, 1993.

[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.

[Mel97] B. Melamed. Grundkonzeption und -implementierung einer Attributkomponente für ein Graphtransformationssystem. Studienarbeit, Technical University of Berlin, FB 13, 1997.

[Mel98] B. Melamed. Design and Realisation of an Attribute Manager for Conditional and Distributed Graph Transformation. Master's thesis, Technical University of Berlin, FB 13, 1998.

[MH01a] André Marburger and Dominikus Herzberg. E-CARES research project: Understanding complex legacy telecommunication systems. In *Proc. 5th European Conference on Software Maintenance and Reengineering*, pages 139–147, Lisbon, Portugal, 2001. IEEE Computer Society Press.

[MH01b] Mark Minas and Berthold Hoffmann. Specifying and implementing visual process modeling languages with DiaGen. In Ehrig et al. [EEP01]. 13 pages.

[MH02] André Marburger and Dominikus Herzberg. E-CARES Research Project: Utilizing Dynamic Information. In *Proc. 3rd Workshop Software Reengineering (WSR 2001)*, pages 47–49. Fachberichte Informatik Universität Koblenz, 2002.

[MM98] J. Meseguer and U. Montanari. Mapping tile logic into rewriting logic. In F. Parisi-Presicce, editor, *Proc. 12th Workshop on Recent Trends in Algebraic Development Techniques (WADT'97)*, volume 1376 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 1998.

[MPPR02] C. Martin-Vide, A. Păun, G. Păun, and G. Rozenberg. Membrane systems with coupled transport: Universality and normal forms. *Fundamenta Informaticae*, 49:1–15, 2002.

[MS99] Manfred Münch and Andy Schürr. Leaving the visual languages ghetto. In *Proc. IEEE Symposium on Visual Languages (VL'99)*, pages 148–155, Los Alamitos, CA, 1999. IEEE Computer Society Press.

[MS00] Manfred Münch and Andy Schürr. Generic and polymorphic graph transformations: Supporting the definition of visual languages. In Corradini and Heckel [CH00], pages 459–466.

[MSW00] Manfred Münch, Andy Schürr, and Andreas Winter. Integrity constraints in the multi-paradigm language PROGRES. In Ehrig et al. [EEKR00], pages 338–351.

[MT98] U. Montanari and C. Talcott. Can actors and π-agents live together? In *Proc. 2nd Workshop on Higher Order Operational Techniques in Semantics (HOOTS'98)*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1998.

[Mün00a] Manfred Münch. Formal specification of dataflow languages with graph grammars. In *ARGESIM Report*. 2000.

[Mün00b] Manfred Münch. Programmed graph rewriting system PROGRES. In Nagl et al. [NSM00], pages 441–448.

[Mün02] Manfred Münch. *Generic Modelling with Graph Rewriting Systems*. PhD thesis, Technical University of Aachen, 2002. To appear.

[NB00] M. Niemann and R. Bardohl. Tool-based Specification of Visual Languages and Graphic Editors. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 456–470. Springer, 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, 2000.

[NZ00] Jörg Niere and Albert Zündorf. Using FUJABA for the development of production control systems. In Nagl et al. [NSM00], pages 181–192.

[PEB00] J. Padberg, C. Ermel, and R. Bardohl. Rule-Based and Visual Model Evolution using GenGEd. In Corradini and Heckel [CH00].

[PGH00] Julia Padberg, Maike Gajewsky, and Kathrin Hoffmann. Incremental development of safety properties in Petri net transformations. In Ehrig et al. [EEKR00].

[Plu97] Detlef Plump. Simplification orders for term graph rewriting. In I. Prívara and P. Ružička, editors, *Proc. Mathematical Foundations of Computer Science (MFCS'97)*, volume 1295 of *Lecture Notes in Computer Science*, pages 458–467. Springer, 1997.

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

[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 Ehrig et al. [EEKR99], chapter 1, pages 3–62.

[Plu02] Detlef Plump. Essentials of term graph rewriting. In Bauderon and Corradini [BC02a].

[PR02] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 2002. To appear.

[PU02] J. Padberg and M. Urbasek. Rule-based refinement of Petri nets: A survey. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Advances in Petri Nets: Petri Net Technologies for Modeling Communication Based Systems*, Lecture Notes in Computer Science. Springer, 2002. To appear.

[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)*, volume 1783 of *Lecture Notes in Computer Science*, pages 304–318. Springer, 2000.

[Rad00a] A. Radermacher. *Tool support for the distribution of object-based applications*. Dissertation, RWTH Aachen, 2000.

[Rad00b] Ansgar Radermacher. Distribution of Applications with Graph Transformation Tools. In Ehrig et al. [EEKR00].

[Rad00c] Ansgar Radermacher. DiTo - a distribution tool based on graph rewriting. In Nagl et al. [NSM00], pages 465–472.

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

[Rud97] Michael Rudolf. Konzeption und Implementierung eines Interpreters für attributierte Graphtransformation. Master's thesis, Technical University of Berlin, FB 13, 1997.

[Rud00] Michael Rudolf. Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching. In Ehrig et al. [EEKR00], pages 238–251.

[Sch97] Andy Schürr. BDL - a nondeterministic data flow programming language with backtracking. In *Proc. Visual Languages '97*, pages 394–401, 1997.

[Sch00] Ansgar Schleicher. Formalizing UML-based process models using graph transformations. In Nagl et al. [NSM00], pages 341–358.

[Sch02] Ansgar Schleicher. *Roundtrip Process Evolution Support in a Wide Spectrum Process Management System*. Deutscher Universitäts-Verlag, Wiesbaden, 2002. To appear.

[SW97] A. Schürr and A.J. Winter. Formal definition and refinement of UML's module/package concept. In J. Bosch and S. Mitchell, editors, *Object-Oriented Technology — ECOOP '97 Workshop Reader*, volume 1357 of *Lecture Notes in Computer Science*, pages 211–215, Berlin, 1997. Springer.

[SW98] A. Schürr and A.J. Winter. Formal definition of UML's package concept. In M. Schader and A. Korthaus, editors, *The Unified Modeling Language — Technical Aspects and Applications*, pages 144–159. Physica-Verlag, Heidelberg, 1998.

[SW00] A. Schürr and A.J. Winter. UML Packages for PROgrammed Graph REwriting Systems. In Ehrig et al. [EEKR00].

[SWZ96] A. Schürr, A.J. Winter, and A. Zündorf. Developing Tools with the PROGRES Environment. In M. Nagl, editor, *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, volume 1170 of *Lecture Notes in Computer Science*, chapter 3.6. Springer, 1996.

[SWZ99] Andy Schürr, Andreas J. Winter, and Albert Zündorf. PROGRES: Language and Environment. In Ehrig et al. [EEKR99], pages 487–550.

[Tae99a] 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'99), June 7 - 10, 1999, Nancy, France.* IEEE Computer Society, 1999.

[Tae99b] G. Taentzer. Distributed graphs and graph transformation. *Applied Categorical Structures*, 7(4):431 – 462, 1999.

[Tae01] G. Taentzer. Towards common exchange formats for graphs and graph transformation systems. In Ehrig et al. [EEP01].

[Tae02a] G. Taentzer. A Visual Modeling Framework for Distributed Object Computing. In B. Jacobs and A. Rensink, editors, *Proc. Formal methods for open object-based distributed systems V (FMODS'02).* Kluwer Academic Publishers, 2002.

[Tae02b] G. Taentzer. Visual modeling of distributed object systems by graph transformation. In Bauderon and Corradini [BC02a].

[TE00] G. Taentzer and H. Ehrig. Semantics of distributed system specifications based on graph transformation. In *GI Workshop "Rigorose Entwicklung software-intensiver Systeme", Berlin*, 2000.

[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.

[TGM00] Gabriele Taentzer, Michael Goedicke, and Torsten Meyer. Dynamic Change Management by Distributed Graph Transformation: Towards Configurable Distributed Systems. In Ehrig et al. [EEKR00], pages 179–193.

[TK97] G. Taentzer and M. Koch. Distributing attributed graph transformation. In *First European Workshop on "General Theory of Graph Transformation Systems", Bordeaux, Oct. 8-10,1997*, 1997.

[TKH+01] I.J. Timm, P. Knirsch, O. Herzog, H.K. Tönshoff, and P.-O. Woelk. Mass Customization als Chance für KMU – Kooperative Agenten für die Informationslogistik. In H.-J. Sebastian and T. Grünert, editors, *Logistik Management – Supply Chain Management und e-Business, Aachen*, pages 401–409. B.G. Teubner Verlag, 2001. In German.

[Tsi00] A. Tsiolakis. Consistency Analysis of UML Class and Sequence Diagrams based on Attributed Typed Graphs and their Transformation. Technical Report 2000/3, Technische Universität Berlin, Department of Computer Science, March 2000.

[TWK+01] I.J. Timm, P.-O. Woelk, P. Knirsch, H.K. Tönshoff, and O. Herzog. Flexible mass customisation: Managing its information logistics using adaptive co-operative multiagent systems. In K.S. Pawar and M. Muffatto, editors, *Logistics and the Digital Economy. Proc. 6th Int. Symposium on Logistics, Salzburg, Austria*, pages 227–232, 2001.

[Ver02] N. Verlinden. *Process Semantics for Embedding-Based Graph Transformation Systems*. PhD thesis, University of Antwerp - UIA, dept. of Math. and Comp. Sci., 2002.

[vG99] John van Groningen. Optimizing Recursive Functions Yielding Multiple Results in Tuples in a Lazy Functional Language. In P. Koopman and C. Clack, editors, *Proc. 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.

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

[WER01] H. Weber, H. Ehrig, and W. Reisig, editors. *2nd Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, Berlin, Germany, Sept. 2001. Researcher Group Petri Net Technology, Fraunhofer Gesellschaft ISST.

[Wes99a] Bernhard Westfechtel. Graph-based product and process management in mechanical engineering. In Ehrig et al. [EEKR99], pages 321–368.

[Wes99b] Bernhard Westfechtel. *Models and Tools for Managing Development Processes*, volume 1646 of *Lecture Notes in Computer Science*. Springer, 1999.

[Wol98] Dietmar Wolz. *Colimit Library for Graph Transformations and Algebraic Development Techniques*. PhD thesis, Technical University of Berlin, FB 13, 1998.

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