Engels, Oberweis, Zündorf (Eds.): Modellierung 2001
LNI P-1, ISBN 3-88579-330-X, pp. 38-48, ©Springer 2001.

Modeling Design Knowledge on Structure[*]

Benno Stein André Schulz

Dept. of Computer Science / Knowledge-Based Systems
University of Paderborn, 33095 Paderborn, Germany
Email: {stein,aschulz}@upb.de

Abstract This paper is on the modeling of design knowledge. It introduces the concept of design graph grammars, which is an advancement of classical graph grammar approaches. Design graph grammars, as proposed here, provide an efficient concept to create and to manipulate structure models of technical systems. This is interesting from two points of view.

Firstly, within many existing tools for design support structural dependencies are represented and processed in a proprietary way. Here, design graph grammars possess the flexibility to model even very specific kinds of domain knowledge while still providing a broadly understood semantics.

Secondly, structure models are close to the mental model level of human designers. Design graph grammars concentrate on this level, excluding involved underlying behavioral aspects. This may entail the risk of an oversimplification, but gives design graph grammars the potential to be used within the creative parts of the human design process.

1 Introduction

Design deals with the creation of new artifacts. Beyond doubt, most design processes require a lot of domain knowledge, a lot of experience, or creativity, and the support of a design process by means of a computer is a long standing wish (BC83; BC89; Ton87).

In this paper we present a new idea to realize design support within engineering domains. Central element is a formalism, called *design graph grammar*, which provides powerful means to specify design knowledge for structure models: By applying graph transformation rules

- a design—say, its underlying structure model—can be analyzed,
- an incompletely and coarsely defined design can be completed and refined, and
- even the reformulation of a design specification with respect to another paradigm is possible.

The paper is organized as follows. The remainder of this section introduces the standard design cycle and motivates the use of graph grammars to solve complex design tasks. Section 2 develops the concept of design graph grammars; in particular, theoretical connections to the most important graph grammar approaches are discussed. Section 3 and 4 then sketch out an application each, in order to demonstrate both the usability and the expressiveness of our ideas.

[*]Supported by DFG grants PA 276/23-1 and KL 529/10-1

1.1 On Design

Gero, renowned for his research in the field of design, defines the term design as follows (Ger90, pg. 28).

> *"The metagoal of design is to transform requirements, generally termed function, which embody the expectations of the purpose of the resulting artifact, into design descriptions."*

The purpose of a design process is the transformation of a complex set of functionalities, D, (the demands) into a design description, M, (a model of the new system):

$$D \longrightarrow M$$

"$\longrightarrow$" stands for some transformation, the model M specifies the system's entire set of components with their relations. The transformation must guarantee that the system being described is able to generate the set D of demands. Due to the complexity and the diversity of a design process no universal theory of design can be stated, i. e., in the very most cases no direct mapping between the elements $d \in D$ and the objects $o \in M$ does exist.

Working on a design problem means to balance two behavior sets, the intended or expected behavior, B_e, and the model behavior, B_M. B_e can directly be derived from a designer's understanding for D, whereas B_M is the result of an analysis of M that may enclose complex model formulation and simulation tasks.

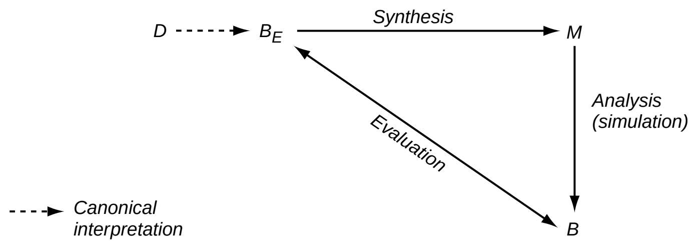


Figure 1: An illustration of Gero's widely-accepted model of the human design process.

Figure 1 illustrates the design process: The expected behavior B_e controls the synthesis of the model M, which in turn yields the model behavior B_M. Within the evaluation phase the two behavior sets are compared to each other; the resulting information serves as input for a new synthesis step.

1.2 The Design Process Reviewed

According to Stein (Ste95), the most creative part of a design process is the definition of the system structure in the form of a structure model. The selection and parameterization, say, the configuration of the components is still demanding but less difficult than structure design. Figure 2 illustrates this view. Here, M_S and M_B designate the structure and behavior models, repectively, and together M_S and M_B make up the model M.

Structure models are very close to the mental model of the human designer (Wal95). Typical examples are circuit diagrams, piping flow charts, bond diagrams, block diagrams, or signal flow graphs (Cel91).

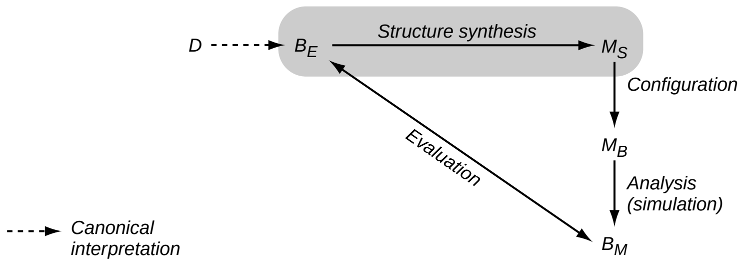


Figure 2: Within the design of complex systems two stages can be distinguished (Ste95): The creation of a structure model, M_S, and the concretization of M_S in the form of a behavior model M_B. The shaded area indicates the places that are predestined for the use of graph grammars.

Actually, the diverse structure models just mentioned represent special forms of labeled graphs.

Definition 1 (Labeled Graph) *A labeled graph is a tuple $G = \langle V_G, E_G, \sigma_G \rangle$ where V_G is the set of nodes, $E_G \subseteq V_G \times V_G$ is the set of directed edges, and σ_G is the label function, $\sigma_G : V_G \cup E_G \rightarrow \Sigma$, where Σ is a set of symbols, called the label alphabet.*

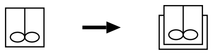
Notation: (v_1, v_2, l) represents a directed edge with tail v_1, head v_2 and label l.

With respect to the generation, the modification, and the analysis of labeled graphs the use of graph grammars has been a proven tool (Roz97; EEKR99; EKMR99; SWZ95; RS95; vSP98). The following section dwells upon this subject.

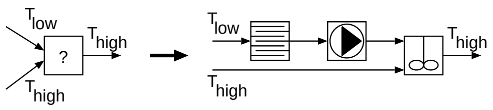
2 Design Tasks and Graph Grammars

The design of a system involves a variety of tasks that pertain to its structure and behavior (cf. Ste95). The solution of these tasks requires that operations of varying complexity are executed. Speaking of manipulation within models represented by labeled graphs, the operations required can be classified as follows:

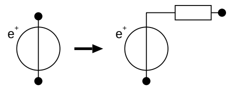
- *Manipulation of Attributes.* Change of specific node and edge attributes. The transformation shown below changes the type of a node from “mixer” to “mixer with built-in heat transfer unit”.



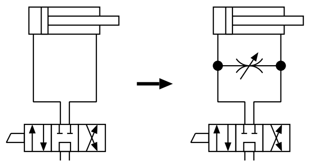
- *Context-free Manipulation.* Insertion and deletion of single nodes and edges in a model. Transformations of this type change the overall structure, as in the following figure, but do not exploit context information.



The transformation depicted below replaces an ideal voltage source by a resistive voltage source.



- *Context-sensitive Manipulation.* Insertion and deletion of sets of nodes and edges, e. g., for model optimization. The following figure shows a context-sensitive transformation in the domain of hydraulics.



The operations delineated above represent transformations on graphs as performed by graph grammars and can be seen as graph transformation rules of the form *target graph → replacement graph*. However, existing graph grammar concepts, although fitting for many tasks, fail to provide the necessary expressiveness to cope with design tasks in general.

2.1 Inadequacy of Classical Graph Grammars

The world of graph grammars is divided into two inherently different approaches: The *connecting approach* and the *gluing approach* (see Roz97, pp. 3–4). The connecting approach is a node-centered concept that has given rise to numerous graph grammars usually called *node replacement* graph grammars. The gluing approach, on the other hand, is a hyperedge-centered approach on which various hypergraph grammars are based.

The most prominent representative of the connecting approach is the *neighborhood controlled embedding (NCE) graph grammar* family[1] (Eng89). NCE graph grammars perform graph transformations on labeled graphs; a graph transformation step is based on node and edge labels, which are used both to increase the discerning power and as some sort of simple context, and does not require any application conditions to hold. Each graph transformation rule also has a set of instructions that dictate the embedding of the transformation, which is done by creating new edges for connection purposes.

Hyperedge replacement (HR) grammars represent the most popular grammar family following the gluing approach. HR grammars replace hyperedges, which are identified through labels, by hypergraphs. Unlike NCE grammars, they lack embedding instructions, which are not necessary here: The embedding is done by mapping the attachment nodes of hyperedges to external nodes of the host hypergraph, which are "glued" together.

Apart from the above described grammars there also exist hybrid approaches that try to combine the features of the connecting and the gluing approach. A widely known approach is the *handle hypergraph (HH) grammar* (CER93; Roz97), which rewrites handles, e. g., hyperedges together with their attachment nodes. The embedding is performed according to the connecting approach. Other hybrid approaches work similarly (cf. KJ96; KH96).

[1] In the literature it is often distinguished between NCE, eNCE, dNCE and edNCE graph grammars. For the sake of simplicity, we refrain from doing so here.

The above described graph grammars are able to handle many tasks, yet they still lack within the following aspects:

- *Missing Requirements*. The design of technical systems often requires that operations be only applied if a certain context is present. The concept of context as an application condition[2] is a requirement for design tasks but is only partially supported by the grammars listed above, which allow only a trivial context based on incident edges.
- *Convenience*. The classical grammars lack some needed features like deletion rules, gap bridging, compact rule formulation etc. These capabilities are also necessary for the solution of design tasks and can only be simulated with classical grammars by means of complex and clumsy rules.

2.2 Design Graph Grammars

Due to the mentioned reasons (and other requirements discussed in (SS00)), the development of a more fitting mechanism—the design graph grammar—becomes a worthwhile undertaking. Since design tasks are usually device-oriented from a human point of view, the design graph grammar should be node-centered. Furthermore, it should provide at least the same descriptive power as the classical NCE grammars and some features of HR grammars, such as the handling of context. Additionally, design graph grammars should come along with further functionality, e. g., to allow for deletion operations. Figure 3 shows the relation of design graph grammars to the classical grammars with respect to the underlying concepts.

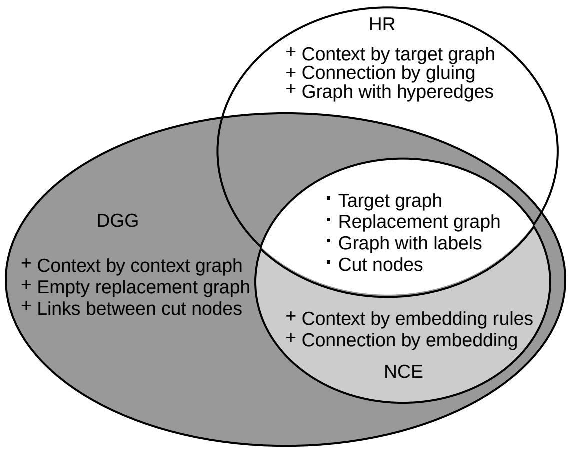

HR
+ Context by target graph
+ Connection by gluing
+ Graph with hyperedges
DGG
+ Context by context graph
+ Empty replacement graph
+ Links between cut nodes
· Target graph
· Replacement graph
· Graph with labels
· Cut nodes
+ Context by embedding rules
+ Connection by embedding
NCE

Figure 3: Relation of design graph grammars to classical grammars with respect to underlying concepts.

The following definition introduces our concept of *design graph grammars* formally. A detailed presentation of design graph grammars and related terms can be found in (SS00).

[2]There is some confusion about the term "context-sensitive" in the literature. E. g., graph replacement grammars, the natural extension to node replacement grammars, are often said to be context-sensitive, although there is no context given as an application condition.

Definition 2 *(Design Graph Grammar) A context-sensitive design graph grammar is a tuple $\mathcal{G} = \langle \Sigma, P, s \rangle$ where*

- *Σ is the label alphabet used for nodes and edges*[3]*,*
- *P is the finite set of graph transformation rules and*
- *s is the initial symbol.*

The graph transformation rules of the set P have the form $\langle T, C \rangle \rightarrow \langle R, I \rangle$, where

- *$T = \langle V_T, E_T, \sigma_T \rangle$ is the target graph to be replaced,*
- *C is a supergraph of T called the context,*
- *$R = \langle V_R, E_R, \sigma_R \rangle$ is the possibly empty replacement graph, and*
- *I is the set of embedding instructions for the replacement graph R.*

  The semantics of a rule $\langle T, C \rangle \rightarrow \langle R, I \rangle$ is as follows: Firstly, a matching of the context C is searched within the host graph. Secondly, an occurrence of T within the matching of C along with all incident edges is deleted. Thirdly, an isomorphic copy of R is connected to the host graph according to the semantics of the embedding instructions.

  The set I of embedding instructions consists of tuples $((h, t, e), (h, r, f))$, where
  - *$h \in \Sigma$ is a node label and $e \in \Sigma$ is an edge label in the host graph incident to T,*
  - *$t \in \Sigma$ is a label of a node in T, for which the current embedding is being specified,*
  - *$f \in \Sigma$ is another edge label,*
  - *and $r \in V_R \cup V_C$ is a node, where V_C is the set of nodes adjacent to the target graph.*

  An embedding instruction $((h, t, e), (h, r, f))$ is interpreted as follows: If there is an edge with label e connecting a node labeled h with the target node t, then the embedding process will create a new edge with label f connecting the node labeled h with node r.

A context-free graph transformation rule may be written as $T \rightarrow \langle R, I \rangle$.

Remarks. Design graph grammars are direction preserving grammars. Obviously, they are closely related to NCE grammars of the connecting approach and possess similar theoretical properties (cf. SS00). Figure 4 shows the relation between design graph grammars, NCE and HR grammars in terms of their descriptive power.

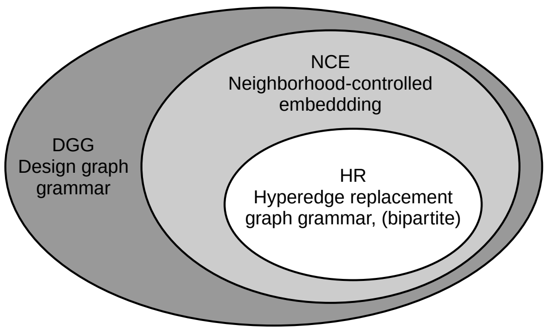


Figure 4: Descriptive power of the different graph grammar concepts.

[3]Labels are used to specify types and as variables for other labels. To avoid confusion, variable labels will be denoted by capital letters, and all other labels with small letters.

2.3 Relationship to Programmed Graph Replacement Systems

Design graph grammars as proposed here shall enable domain experts to formulate design expertise for various design tasks. Design graph grammars result from the combination of different features of the classical graph grammar approaches, while special effort has been spent to keep the underlying formalism as simple as possible.

When comparing design graph grammars to programmed graph replacement systems (PGRS) one should keep in mind that the former is located at the conceptual level while the latter emphasizes the tool character. PGRS are centered around a complex language allowing for different programming approaches. PROGRES, for instance, offers declarative and procedural elements (Sch89) for data flow oriented, object oriented, rule based and imperative programming styles. A direct comparison between PROGRES to the concept of design graph grammars is of restricted use only and must stay at the level of abstract graph transformation mechanisms.

However, it is useful to relate the concepts of design graph grammars to PGRS under the viewpoint of operationalization. PGRS are a means—say: one possibility—to realize a design graph grammar by reproducing its concepts. In this connection PROGRES fulfills the requirements of design graph grammars for the most part. However, PROGRES lacks the design graph grammar facilities for the formulation of context, deletion operations, and gap bridging, which have to be simulated by means of complex rules. Such a kind of emulation may be useful as a prototypic implementation, but basically, it misses a major concern of design graph grammars: Their intended compactness, simplicity, and adaptivity with respect to a concrete domain or task.

The following sections present two applications where design graph grammars have been employed.

3 Application I. Conceptual Design in Chemical Engineering

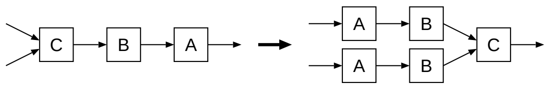
In the domain of chemical engineering the design of a system may encompass various tasks, of which many can be solved by means of structural manipulation. Synthesis of a new design structure may be viewed as the derivation of a “word” of a graph language by a design graph grammar that encodes engineering knowledge (SSK00; SS00). Similarly, structural analysis of a given design may be performed by the same design graph grammar working reversely—this corresponds to solving the membership problem for graph languages (see SS00).

Another design task that can be tackled by structural manipulation is the modification of a given design, which is also a synthesis step. Modifications are necessary to repair or optimize structures of design solutions. The following simplified rules perform optimizations on designs and belong to the rule base of a project that concentrates on modeling and optimization in the food processing industry.

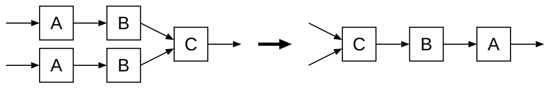
(R_1) Relocation and splitting of a processing chain to achieve better processing properties (reduction of processing time).

$$\begin{aligned}
T &= \langle V_T, E_T, \sigma_T \rangle = \langle \{m, n, o\}, \{(m, n), (n, o)\}, \{(m, C), (n, B), (o, A)\}\rangle \\
R &= \langle V_R, E_R, \sigma_R \rangle = \langle \{t, u, v, w, x\}, \{(t, u), (u, x), (v, w), (w, x)\}, \\
&\quad\ \{(t, A), (u, B), (v, A), (w, B), (x, C)\}\rangle \\
I &= \{((H, C, L), (H, t, L)), ((H, C, L), (H, v, L)), ((H, A, L), (H, C, L))\}
\end{aligned}$$

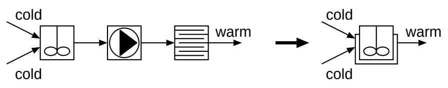
For illustrative reasons we now use the graphical representation of the above rule and omit the formal version.



(R_2) Merging and relocation of processing chains to reduce plant costs (reduction of number of devices).



(R_3) Replacement of a mixer and a heating chain by a mixer with built-in heat transfer unit (reduction of costs).



(R_4) Replacement of a mixer and a heating chain by a static mixer with built-in heat transfer unit (reduction of energy consumption, improvement of hygienic properties).

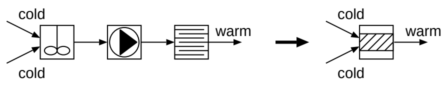


Figure 5 shows how these rules are applied to a design. The first graph in the derivation corresponds to the original state of the design.

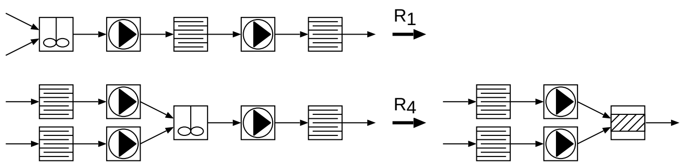


Figure 5: Application of two graph transformation rules to a chemical plant design.

Remarks. These rules have been simplified for presentation purposes here; they represent an excerpt of our rule base that is necessary to build realistic plant designs for the food processing industry. In fact, the final rule base will have between 60 and 80 rules, of which about two thirds will be task-specific. The quality of automatically generated designs can be expected to be acceptable for a human expert if the plant size does not exceed considerably the number of 20 nodes.

4 Application II. Synthesis of Wave Digital Structures

Wave digital structures, abbreviated: WDS, form a particular class of signal flow graphs where the signals are linear combinations of the electric current and flow. WDS establish a concept to reformulate electrical circuits from the electrical u/i-domain into the a/b-wave-domain. This reformulation is bound up with several advantages, which are not discussed in this place but can be found in (Fet86).

When migrating from an electrical circuit towards a WDS, the underlying model is completely changed: The structure model of the electrical circuit, $M_S^{u/i}$, is interpreted as a series-parallel graph with closely connected components and transformed into an adaptor structure, $M_S^{a/b}$.

Figure 6 shows the reformulation of a series-parallel structure tree of an electrical circuit into a corresponding adaptor structure. The nodes labeled by “s” and “p” indicate series and parallel connections in the circuit.

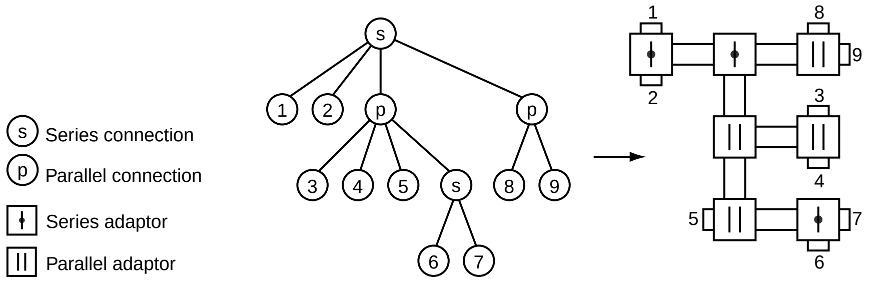


Figure 6: Overview of the mapping $M_S^{u/i} \longrightarrow M_S^{a/b}$.

The mapping $M_S^{u/i} \longrightarrow M_S^{a/b}$ establishes a paradigm shift. Other examples of such paradigm shifts are the migration from relational to object-oriented databases, or the migration from numeric processing to symbolic processing. In terms of models, such a translation is called *model reformulation*.

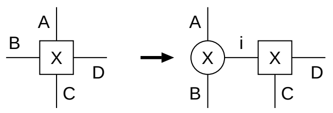
The following design graph grammar[4] performs the model reformulation depicted in Figure 6 for arbitrary structure models $M_S^{u/i}$.

$\mathcal{G} = \langle \Sigma, P, z \rangle$ with $\Sigma = \{z, p, s, i, X, Y\} \cup \mathbf{N}$, z is the initial symbol (can be neglected), and P is the set of graph transformation rules, which are presented in the following.

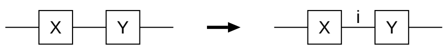
(1) Splitting rule for nodes with more than three edges.

$$\begin{aligned}
T &= \langle V_T, E_T, \sigma_T \rangle = \langle \{a\}, \emptyset, \{(a, X)\} \rangle \\
R &= \langle V_R, E_R, \sigma_R \rangle = \langle \{b, c\}, \{(b, c)\}, \{(b, X), (c, X), (\{b, c\}, i)\} \rangle \\
I &= \{((H, X, A), (H, b, A)), ((H, X, B), (H, b, B)), ((H, X, C), (H, c, C)), \\
&\quad ((H, X, D), (H, c, D))\}
\end{aligned}$$

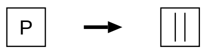
For illustrative reasons we resort to the graphical representation[5] from now on and refrain from using the formal version if appropriate.

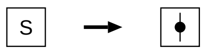


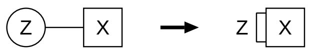
[4]Since the involved transformation is a translation rather than generation, it would be better to speak of graph transformation systems instead of graph grammars.

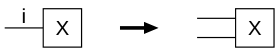
(2) Marking rule for edges. 

The above rules are sufficient to perform the structural transformation required. The following rules belonging to an additional design graph grammar are necessary to change the appearance of the final structure into an adaptor structure as depicted in Figure 6.

(1) Display of a parallel node. 

(2) Display of a serial node. 

(3) Display of a port node. 

(4) Display of node connector. 

5 Summary

The modeling of human design knowledge is a key problem when developing programs that shall automate a given design task all or in part. Design graph grammars, as proposed in this paper, are a novel approach to encode design knowledge in complex engineering tasks: They have been created as an instrument to formulate very different kinds of structure knowledge while still providing a well-defined semantics.

In our working group there is a long tradition in solving design and configuration tasks in engineering domains. The design graph grammar approach introduced here is a result of the analysis of several projects involving structure model manipulation. Two projects, from the domains of chemical and electrical engineering, have been delineated to exemplify the use of design graph grammars.

The development of design graph grammars has just begun, and this concept has yet to be validated through heavy-weight applications from technical domains. Regardless of the actual status, we are confident that design graph grammars will establish a useful contribution for modeling and knowledge representation in the field of engineering design.

References

[BC83] D. C. Brown and B. Chandrasekaran. An Approach to Expert Systems for Mechanical Design. In *Trends and Applications '83*. IEEE Computer Society, NBS, Gaithersburg, MD, 1983.

[BC89] David C. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.

[Cel91] François E. Cellier. *Continuous System Simulation*. Springer, Berlin Heidelberg New York, 1991.

[CER93] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting Hypergraph Grammars. *Journal of Computer and System Sciences*, 46:218–270, 1993.

[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.

[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.

[Eng89] J. Engelfriet. Context-free NCE Graph Grammars. In J. Csirik, J. Demetrovics, and F. Gécseg, editors, *Proc. Fundamentals of Computation Theory*, volume 380 of *Lecture Notes in Computer Science*, pages 148–161. Springer-Verlag, 1989.

[Fet86] Alfred Fettweis. Wave Digital Filters: Theory and Practice. *Proceedings of the IEEE*, 74(2):270–327, February 1986.

[Ger90] John S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.

[KH96] Renate Klempien-Hinrichs. Node Replacement in Hypergraphs: Simulation of Hyperedge Replacement and Decidability of Confluence. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, number 1073 in Lecture Notes in Computer Science, pages 397–411, Berlin, 1996. Springer-Verlag.

[KJ96] Changwook Kim and Tae Eui Jeong. HRNCE Grammars – A Hypergraph Generating System with an eNCE Way of Rewriting. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, number 1073 in Lecture Notes in Computer Science, pages 383–396, Berlin, 1996. Springer-Verlag.

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

[RS95] J. Rekers and A. Schürr. A Graph Grammar Approach to Graphical Parsing. Technical Report 95-15, Department of Computer Science, Leiden University, 1995.

[Sch89] Andy Schürr. Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language. In M. Nagl, editor, *Proc. 15th Intl. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 411 of *LNCS*, pages 151–165. Springer-Verlag, 1989.

[SS00] André Schulz and Benno Stein. On Automated Design of Technical Systems. Technical Report tr-ri-00-218, University of Paderborn, 2000.

[SSK00] André Schulz, Benno Stein, and Annett Kurzok. On Automated Design in Chemical Engineering. In R. J. Howlett and L. C. Jain, editors, *Proc. 4th Intl. Conference on Knowledge-based Intelligent Engineering Systems & Allied Technologies*, pages 261–266. IEEE, september 2000.

[Ste95] Benno Stein. *Functional Models in Configuration Systems*. PhD thesis, University of Paderborn, 1995.

[SWZ95] A. Schürr, A. Winter, and A. Zündorf. Visual Programming with Graph Rewriting Systems. In *Proc. 11th Intl. IEEE Symposium on Visual Languages*. IEEE Computer Society Press, 1995.

[Ton87] Christopher Tong. Towards an Engineering Science of Knowledge-based Design. *Artificial Intelligence in Engineering*, 2(3):133–166, 1987.

[vSP98] Marko van Eekelen, Sjaak Smetsers, and Rinus Plasmeijer. Graph Rewriting Systems for Functional Programming Languages. Technical report, Computing Science Institute, University of Nijmegen, 1998.

[Wal95] J. Wallaschek. Modellierung und Simulation als Beitrag zur Verkürzung der Entwicklungszeiten mechatronischer Produkte. *VDI Berichte, Nr. 1215*, pages 35–50, 1995.