A New Method for Formal Description of 3D Scenes

Vladimír Manda
xmanda@sun.felk.cvut.cz
Pavel Slavík
slavik@cs.felk.cvut.cz

Abstract

In this paper we deal with the problem of representation of 3D scenes. As far as the scene structure is concerned, it contains both objects and their relations. To capture this we propose to use a special graph grammar language. This language allows us to describe the objects in the scene and their properties, as well as the relations. Moreover, we can describe the constraints that can be imposed upon the scene. This particular paper introduces the idea and shows how to represent the objects in a simple way using proposed concepts.

Keywords

graph grammar, modelling, constraints

1 Formal frameworks for the description of graphical objects

A picture is usually considered a set of lines (in the case of vector images) or a set of pixels (in the case of raster images). Lines and vectors in both cases are treated as independent entities. Usually the picture has some formal structure — usually a hierarchical one. This allows for distinguishing certain parts of the picture that can be further decomposed into smaller parts etc. In such a way it is possible to describe the picture by formal means.

The picture can be considered to be a sentence in a language (a picture one). It is possible to define a set of rules that will constitute a grammar (picture grammar) describing a class of pictures [1]. The use of such a grammar will be similar to the use of grammars in the string case — it is possible either to generate pictures that belong to the picture class or to parse an existing picture in order to investigate if the picture belongs to the given class of pictures. There have been many systems developed in the past that used this grammatical approach to create and manipulate pictures [2].

The use of syntactic description of a graphical object has other advantages both from the point of view of the object description and from the point of view of interaction. We should have in mind that the formal description of a graphical object can be very memory consuming — that means that the corresponding data structures can be rather extensive. In many cases we can store the object description in the form of a sequence of numbers of rules used for the derivation of object description.

As graphical objects in general have a complicated nature it is advisable to find a more general framework by means of which would be possible to describe complicated graphical objects. When we speak about line oriented pictures we can come to a conclusion that each picture of this kind can be described in a formal way by means of a graph. We can define a set of rules that will constitute a graph grammar that will define a class of graphs. These graphs can be interpreted as pictures (thus also a creating certain class).

In the case of graph grammars there is defined a set of rules that describes the substitution of a subgraph in a graph by another subgraph. This formalism is widely used in many applications — mostly for the formal description of various techniques used in the field of computer science [3]. The idea to use graph grammar for the formal description of pictures is not a new one — many systems in the past were developed that described certain classes of pictures in this way.

In the more general case we can describe configuration of 2D objects by means of graphs. The objects themselves will be represented as graph nodes and relations between objects can be described by means of edges. This approach allows description of complicated configurations of 2D objects — in case a graph grammar was used these configurations belong to a certain class.

2 Three dimensional applications of graph grammars

In the previous chapter some formalisms that are used for the description of 2D objects were described. The main advantage of these formalisms is the fact that the configuration of the objects and their parts was created in accordance with some rules. The activities performed by the user were fully deterministic and well defined. In such a way the user performed the valid operations only.

The experience gained in the 2D world can be extended into 3D. Also in this case it is necessary to create means that can describe the user's actions in a formal way. The most promising approach is that of graph grammars because of their generality. Usually the grammatical approach is mentioned in the case when a configuration of objects (both 2D and 3D) is created (generated). When using formalism that is powerful enough we can use methods that can describe manipulation with the scene and control the scene consistency at the same time.

Furthermore we will describe a simple system that allows us to define and manipulate 3D scenes that are composed out of cubes the faces of which are parallel to the xy, xz and yz planes. The operations available are as follows:

a) a new cube will be added to the current scene configuration

b) a cube will be removed from the scene

c) a cube can be moved to a new position and its size could be changed

The formalism that will allow us to formalize the above given operations is called PAGG (Programmed Attributed Graph Grammars). This formalism is an extension of the graph grammar formalism that was mentioned in the previous chapter. The general principles of this formalism (PAGG) will be explained in the following sections together with the description of functions that perform desired scene manipulation.

3 Programmed Attributed Graph Grammars

The theory of PAGG was developed at Technical University in Vienna [4]. It was originally used as a framework for the formal description of software. The theory was also successfully applied to the theory of object databases and of neural networks [5].

In this section we present the basic concepts of graph-based data representation as described in [4].

3.1 Preliminary definitions

In this subsection we present the basic definitions. The complete version of the definitions can be found in [4].

Definition 1. (*Scheme*) A scheme is a 7-tuple $S = (V, W, F, A, B, C, D)$, with V being a finite set of node labels, W a finite set of edge labels, $F \subseteq V \times W \times V$, A a set of attribute values for the nodes, B a set of attribute values for the edges, and $C : V \to 2^A$ (where 2^A denotes the powerset of the set A) respectively $D : W \to 2^B$ the domain of allowed attributes for each specific node respectively edge label.

Definition 2. A *graph* over (V, W) is a triple $g = (N, n, E)$, where N is a finite set of *nodes*, $n : N \to V$ is the labelling function for the nodes in N, $E \subseteq N \times W \times N$. Any tuple (u, k, v) with $u, v \in N$ and $k \in W$ can be interpreted as a (*directed*) *edge* from the node u to the node v and labelled by k. The set of all graphs over (V, W) is denoted by $\gamma(V, W)$. An *attributed graph* over (V, W, A, B) is a quintuple $g = (N, n, E, a, b)$, where $g^* = (N, n, E) \in \gamma(V, W)$ is the underlying graph and $a : N \to A$ and $b : E \to B$ are the *attribution functions* assigning attributes of the sets A respectively B to each of the nodes in N respectively to each edge in E. The set of all attributed graphs over (V, W, A, B) is denoted by $\gamma(V, W, A, B)$.

Definition 3. (*Instance*) Let $S = (V, W, F, A, B, C, D)$ be a scheme. An *instance* over S is an attributed graph $g = (N, n, E, A, B) \in \gamma(V, W, A, B)$ such that $(n(u), k, n(v)) \in F$ for each edge $(u, k, v) \in E$ with $u, v \in V$, $k \in W$, as well as $a(v) \in C(n(v))$ for all $v \in N$ and $b((u, k, v)) \in D(k)$ for all $(u, k, v) \in E$.

Definition 4. (*Pattern*) Let $S = (V, W, F, A, B, C, D)$ be a scheme and let $g = (N, n, E, A, B) \in \gamma(V, W, 2^A, 2^B)$. Then g is called a *pattern* over S, if $(n(u), k, n(v)) \in F$ for each edge $(u, k, v) \in E$ with $u, v \in N$, $k \in W$, and, moreover, $a(v) \subseteq C(n(v))$ for all $v \in N$ as well as $b((u, k, v)) \subseteq D(k)$ for all $(u, k, v) \in E$.

3.2 Set of attributed graph productions

In this subsection we describe a special set of graph productions.

Definition 5. Let λ be a special (blank) symbol, and let V, W be alphabets; then we define $V_\lambda := V \cup \{\lambda\}$ and $W_\lambda := W \cup \{\lambda\}$. The set of node productions over (V, W) is defined by $R_N(V, W) := V_\lambda^2$; the set of edge productions over (V, W) is defined by $R_E(V, W) := (V \times W_\lambda \times V)^2$. Any element of $R(V, W) := R_N(V, W) \cup R_E(V, W)$ is called a *graph production.*

An attributed graph production is a pair (p, f), where p is a graph production and f is a function working on the attributes of the node(s) respectively the edge affected by the graph production p. In the following, we list all possible attributed graph productions

(p, f) with their informal description. The formal definitions of applicability conditions and of the resulting attributed graph can be found in [4].

1. $((\lambda, L), C(a))$
   Add a new node u with label L and attribute a. $C(a)$ is a characteristic function that introduces the attribute a.
2. $((K, L), f)$ *with* $f : A \to A$
   Change the label of a node u from K to L and change the attribute according to the function f.
3. $((K, \lambda), \emptyset)$
   Delete a node u with label K, but only if no edges are leaving from or ending in the node u with label K of the graph g.
4. $((\lambda, \lambda), \emptyset)$
   No operation.
5. $((K, \lambda, L; M, k, N), f)$ *with* $f : (A \times A) \to (A \times B \times A)$
   Add an edge with label k between a node u with label K and a node v with label L and change these node labels from K to M and form L to N, respectively, but only if (u, k, v) is not already an edge in g, change the attributes of the nodes u and v according to the attribution function f and assign an attribute to the new edge.
6. $((K, j, L; M, k, N), f)$ *with* $f : (A \times B \times A) \to (A \times B \times A)$
   Change the label of an edge between a node u with label K and a node v with label L from j to k and change these node labels from K to M and from L to N, respectively, but only if (u, k, v) is not already an edge in g, change the attributes of the nodes u and v and of the edge (u, k, v) according to the function f.
7. $((K, j, L; M, \lambda, N), f)$ *with* $f : (A \times B \times A) \to (A \times A)$
   Delete an edge with label j between a node u with label K and a node v with label L and change these node labels from K to M and from L to N, respectively, and change the attributes of the nodes u and v according to the attribution function f.
8. $((K, \lambda, L; M, \lambda, N), f)$ *with* $f : (A \times A) \to (A \times A)$
   Whatever the edges between a node u with label K and a node v with label L are, change these node labels from K to M and from L to N, respectively, and change the attribute values of the nodes u and v according to the attribution function f.

The set of all these attributed graph productions is denoted by $R(V, W, A, B)$.

Definition 6. A *graph controlled* (*attributed graph grammar*) system is a 7-tuple $G = (V, W, A, B, P, p_0, p_f)$ with the following characteristics:

1. V and W are alphabets for labelling the nodes and edges, respectively, and A and B are the sets of attributes for the nodes and edges, respectively.
2. P is a finite set of triples (*rules*) $(r : p, \sigma(r), \varphi(r))$, where r is the label of the production $p \in R(V, W, A, B)$, $\sigma(r) \subseteq Lab(p)$ and $\varphi(r) \subseteq Lab(p)$, where $Lab(p)$ denotes the labels of the productions in P.
3. p_0 is the initial label (initial rule).
4. p_f is the final label (final rule).

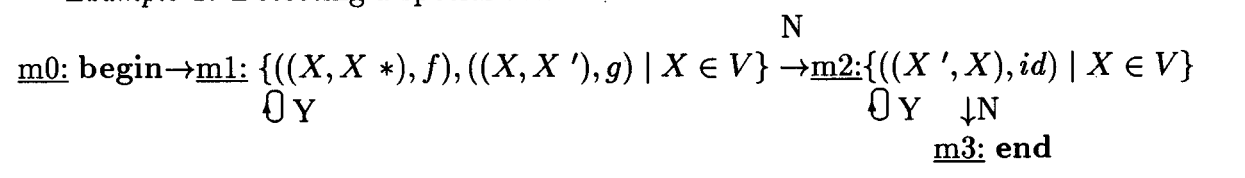
In the following we are interested in graph controlled systems as rewriting systems on attributed graphs suited for the specification of operations on data represented by attributed graphs.

A graph controlled system can be described by a control diagram, where each rule $(r : p, \sigma(r), \varphi(r))$ is represented by $r : p$ and edges (represented as arrows) leaving this node of the diagram; these edges are labelled by **Y**(es) and **N**(o) and end up in the nodes representing the rules in $\sigma(r)$ and $\varphi(r)$, respectively. The production that will follow after $r : p$ is the one with label $\sigma(r)$ if the production $r : p$ is applicable, otherwise $\varphi(r)$ follows. If $\sigma(r)$ or $\varphi(r)$ is r, then we obtain a *loop*. Moreover we use **begin** and **end** to indicate the initial and the final rule, respectively.

To allow well-arranged control diagrams, we use modular concept. A module is a control diagram, that begins with one/more **begin** and ends with at least one **end** rule. Modules are identified by name(s) of their **begin** rules. Modules can be linked via these special rules.

A special feature is *marking*. This concept is included in the basic theory of PAGG and is used very much, so we mention it separately. Some symbols (e.g. $*$) of the alphabet of labels are reserved as *marking symbols*. The marking concept is then used for example as follows: one module searches for a particular node, then it marks it with a marking symbol. Then the following module changes the label, computes the attribution functions, and removes the marking symbol.

Example 1. Detecting a special attribute.



$f(x) = \underline{\text{if}}\ x = c\ \underline{\text{then}}\ x;\ g(x) = \underline{\text{if}}\ x \neq c\ \underline{\text{then}}\ x$

□

Note: by ↻ we denote a loop.

Using the graph controlled system represented by the control diagram in Example 1 we can detect every node the attribute of which coincides with a given constant c. These nodes are marked with a star, because the attribution function f associated with the graph productions $(X, X *)$, $X \in V$, can be applied at the rule m1, whereas the other nodes are marked with a prime at the rule m1, when the attribution function g associated with the graph productions $(X, X ')$, $X \in V$, has to be applied at the rule m1, and are relabelled by the rule m2.

For more detailed definitions and descriptions the reader is referred to [4].

4 Using PAGG for 3D scene description

As we have mentioned in the first section, the geometry–modelling problems are often graph problems. Thus the theory of PAGG is the most natural framework for representing and describing manipulation of graph scenes. In this section we show how we map geometrical properties onto graph properties.

The basic concept is obvious. Each particular object in the scene and its properties are represented as a graph node and its attributes, respectively, and the relations between objects and their properties are represented by graph edges and their attributes, respectively. Thus there is a direct analogy between the scene and the attributed graph.

Fig. 1 shows a simple example of mapping a scene to an attributed graph. The scene consists of cubes with their properties (for detailed description see section 5).

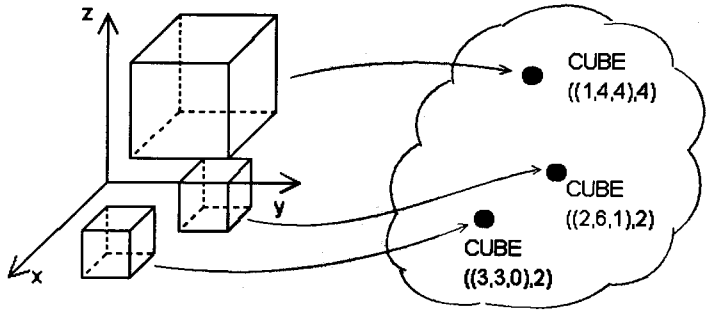


Fig. 1: Mapping a scene to an attributed graph

The process of building and modifying the scene structure (i.e. adding and deleting new objects, adding and deleting object relations) directly corresponds to the attributed graph productions (i.e. add and delete a node, add and delete an edge). Modifications of object properties correspond to the graph productions that change the attributes of graph nodes and edges.

As can be observed, there is a direct correspondence between building the scene and building the attributed graph. The type of a particular attributed graph production can be directly derived from the type of operation on the scene structure. Deriving the attribution functions themselves is less trivial. In most cases the attribution function is the identical function, that does not change the attributes, or a simple function working on the attributes. More problems come with the 'non effective' attributed graph production. We call them 'service productions'. These productions usually do not change the graph structure, but only help us to deal with the global configuration of the attributed graph. The reason is that the attributed graph productions are strictly local operations. We have no means to manipulate the attributed graph in a global way. The global operations we need very often are such as searching for a particular node, changing attributes of a set of nodes, etc. In these cases we use a marking technique (see Example 1) together with carefully designed attribution functions.

Other features, that help us to deal with the complexity of control diagrams are modularity and parameterization. A parameterized module would for example do searching for a node with properties given to it as parameters. Thus we can effectively reuse the code and simplify the resulting control diagram.

In the following section we specify our case-study and give some examples that make the previously shown guidelines clear.

5 The case-study

To keep the control diagrams simple and readable, we have chosen simple objects to be in the scene and only some of their properties are used.

We define the scene and its properties as follows:

1. Scene objects: cubes with their faces parallel to co-ordinate planes.
2. Scene objects' properties: position of the vertex with the lowest co-ordinates, length of the edge; we denote $((x_0, y_0, z_0), a_0)$.
3. We only represent the scene structure, i.e. the structure of objects in the scene.

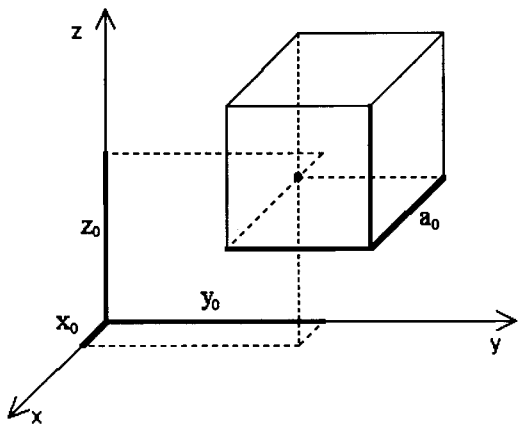


Fig. 2: Cube attributes

Then, step-by-step, we derive a particular framework for the case-study. We must specify a customized scheme (we use prefix 'Scene') first.

Definition 7. (*Scene scheme*) A scene scheme is derived from a scheme (see Definition 1) as follows:

1. we specify the sets of labels:
   $M = \{*\}^a \cup \{\hat{}\}^b \cup \{\#\}^c \cup \{\$\}^d \cup \{@\}^e, a, b, c, d, e \in R$, is a set of marking symbols,
   $V = \{CUBE\ m \mid m \subseteq M\}$ is a set of all object labels, where $CUBE$ denotes the type of object (all of them are of type $CUBE$ (i.e. cubes)) and m is any of all possible combinations of marking symbols,
   $W = \emptyset$ is a set of all relation labels (we have no relations yet).

2. we specify all the allowed combinations (node label, edge label, node label):
   $F = \{\forall(X, \lambda, Y) \mid X, Y \in V\}$ (there can be any combination of two node labels — we do not have any relations defined, anyway)

3. we specify the sets of all allowed attributes:
   $A = \{\forall((x, y, z), a) \mid x, y, z, a \in R\}$ is the set of allowed properties of objects,
   $B = \emptyset$ (we have no attributes for edges).

4. we specify the domains of allowed attributes for nodes and edges
   $C = \{\forall(CUBE\ m, ((x, y, z), a)) \mid x, y, z, a \in R, m \subseteq M\}$ each cube has its position and size assigned,
   $D = \emptyset$ (no attributes are allowed to the edges).

In the following examples we show how the basic operations can be specified.

Example 2. Adding a new cube with properties $((x, y, z), a)$.

a0: **begin** [add cube $(((x, y, z), a))$]

↓

a1: $\{((\lambda, CUBE\ *), C(((x, y, z), a))\}$

↓

a2: **end** [add cube]

□

Example 2 shows a parameterized module that adds a cube to the scene. The function C is a characteristic function that assigns cube properties (given by parameters) to the attributes of a newly created node labelled $CUBE\ *$. It is marked with a star to help the operations that will follow this operation. What happens in the scene and in corresponding attributed graph is shown in Fig. 3.

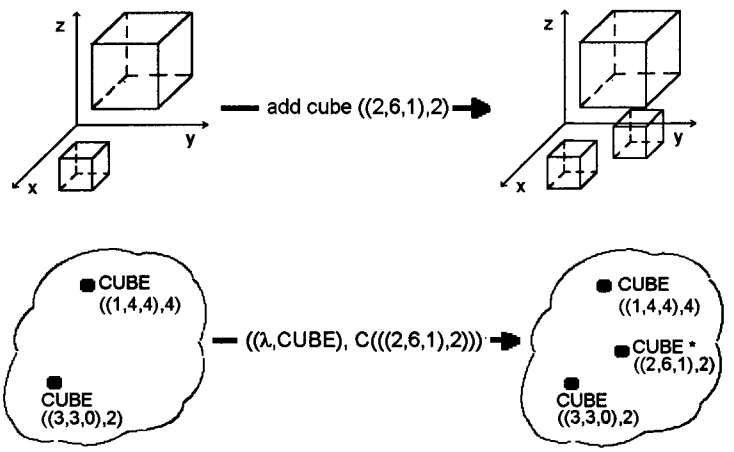


Fig. 3: Adding a cube to the scene

Example 3. Searching for a cube with given properties.

s0: **begin** [search cube $(((x, y, z), a))$]

↓

s1: $\{((X, X\ \hat{}), f), ((X, X\ \#), g) \mid X \in V\}$

↓N ↺Y N

s2: $\{((X\ \#, X), id) \mid X \in V\} \rightarrow$s3: **end** [search cube]

↺Y

id is the identity function ,

$f(((x_0, y_0, z_0), a_0)) =$ if $((x_0, y_0, z_0), a_0) = ((x, y, z), a)$ then $((x_0, y_0, z_0), a_0)$

$g(((x_0, y_0, z_0), a_0)) =$ if $((x_0, y_0, z_0), a_0) \neq ((x, y, z), a)$ then $((x_0, y_0, z_0), a_0)$

□

This example is directly derived from Example 1. Only the attribution functions are specified according to the attributes from scene scheme. In the first loop (s1) the nodes which meet requirements are marked with $\hat{}$ and the other nodes with #. In the second loop (s2) the latter nodes are unmarked. This loop does the unmarking while there are any #-marked nodes.

Fig. 4 shows an example of searching for a particular cube.

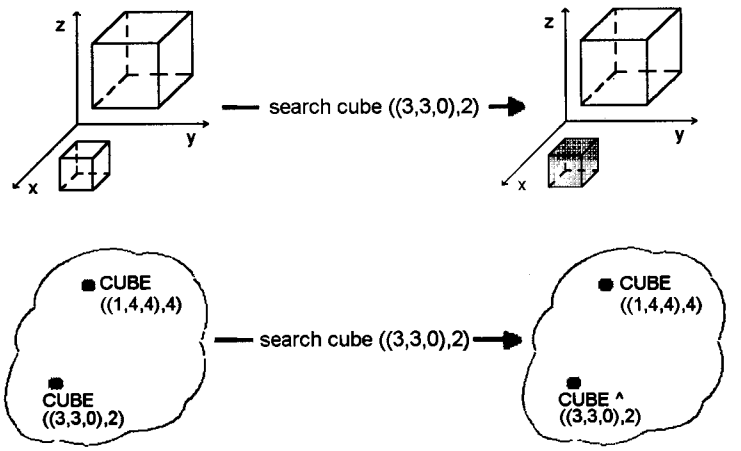


Fig. 4: Searching for a cube

Example 4. Changing properties of given cube(s).

ch0: **begin** [change cubes $(((x_1, y_1, z_1), a_1), ((x_2, y_2, z_2), a_2))$]

↓

ch1: search cube $(((x_1, y_1, z_1), a_1))$

↓

ch2: $\{((X\ \hat{},X),h) \mid X \in V\}$

↓N ↻Y

ch3: **end** [change cubes]

$h(((x_0, y_0, z_0), a_0)) = ((x_2, y_2, z_2), a_2)$

□

This example shows how we use modules and parameters. If we want to change the attributes of a given cube(s), we must find the cube(s) first. Then we go through the marked nodes and change the attributes to a new value.

In the same way we can make a whole set of operations we need.

Adding relations between objects will introduce edges to the attributed graph. We will concentrate on this in our future work.

In this chapter some introductory examples have been shown. To build a complete set of operations, a good modular concept should be developed for particular applications. Only then can the diagrams be kept simple.

Anyway, it has not been said how complex the graph controlled system is. For example we do not say how the production ch2 in Example 4. is performed. We leave it on the underlying layer (e.g. the PAGG interpreter or compiler).

6 Conclusion, future work

In the previous sections we have shown how to map an arbitrary scene to an attributed graph in a simple way. Moreover, we have shown a simple tool for manipulating the objects in the scene. The control diagrams can be derived from the guidelines we should use when building the scene 'by hand'. Deep theory background together with parameterization and modularization makes the approach systematic.

One of the great benefits is that the user can use the modules without detailed knowledge what happens inside — the module could for example do some consistency checking

of the scene. An example is: the user wants to add a cube to the scene, but the system inside decides if he/she is performing an allowed action (maybe the cube size exceeds some limit). This is allowed by the concept: there is a kind of database engine (graph engine) in the background that does it all for us. Thus the access to the data is transparent and can be independent from the computational platform. This concept is similar to encapsulation in object-oriented programming.

The problem becomes more interesting when the relations are introduced. By adding them to the scene, many control diagrams become more complicated by having to fulfil the constraints the relations imply. A reasonable relation can be for example 'constant distance between two objects in the scene'. For any such relation a label and attributes must be specified. Then the relation constraints must be built into existing control diagrams as well as new (relation-specific) control diagrams must be designed. We will concentrate on this in our future work and we will also do more testing of proposed concepts.

7 References

[1] Rosenfeld, A.: *Quadtree Grammars for Picture Languages*, IEEE, Trans. on Systems, Man and Cybernetics, May–June 1982, pp. 401–405.

[2] Slavík, P.,: *Syntactic Methods in Computer Graphics*, Proc. of Eurographics 83, North Holland 1983, pp. 133–142.

[3] Engels, G., et al.: *Graph Grammar Engineering — A software Specification Method.* In: Ehrig, H., et al.: Graph Grammars and Their Applications to Computer Science, LNCS 291, Springer, 1987.

[4] Freund, R., Stary, C.: *Formal Software Specification Using Graph Rewriting Systems.* Technical report, University Magdeburg, 1994.

[5] Bischof, H., Freund, R., Haberstroh, B.: *Structure Optimization of Neural Networks by Graph Rewriting Systems.* Technical report, TU Vienna, 1995.