Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 6 pages

Visual Language Parsing in GenGEd [1]

R. Bardohl, T. Schultzke, G. Taentzer

Dept. of Computer Science
TU Berlin, Germany

Abstract

GenGEd supports the visual specification of visual languages and the generation of syntax-directed editors. However, syntax-directed editing is not always desired by the user. Therefore we extended GenGEd by parsing facilities which allow for free editing as well.

1 Introduction

Syntactical definition of visual languages (VLs) and VL-parsing are difficult problems due to the absence of an easy to use and efficiently parsable standard syntax definition formalism. Most proposals published up to now rely on context-free grammar rules, i.e., they allow replacement of a single nonterminal in the left-hand side. Using these approaches it is not always possible to define the VL in mind. Therefore, context-sensitive graph grammars have been proposed, e.g., in the form of *Layered Graph Grammars (LGG)* [11]. LGG rules are allowed to delete and create several elements and relations, represented as vertices and edges.

Unfortunately, LGGs are still not convenient enough to define a VL in general because of at least missing *Negative Application Conditions (NACs)* and further conditions for rules. Therefore, a new form of LGGs, called *Contextual Layered Graph Grammars (CLGGs)* was developed [2] which support vertex embedding, NACs, and complex predicates. This approach includes the definition of layering conditions guaranteeing termination of the parsing process. Furthermore, static analysis techniques like critical pair analysis [10,9,12] are available which can be exploited to identify a maximum set of rules which may be parsed without any need for backtracking.

VLCC [3] and DiaGen [8] use restricted context-sensitive rules to parse VLs. This kind of rules is successfully applied to rather simple languages but

[1] Research partially supported by the German Research Council (DFG), and the projects APPLIGRAPH (ESPRIT Basic Research WG) and GRAPHIT (CNPq and DLR).

might be tricky to use in more complex cases. CLGGs are related to *Reserved Graph Grammars (RGGs)*, another restricted and modified form of LGGs [13]. RGGs offer some kind of embedding mechanism, too, but do not support the definition of predicates (however, not used in GenGEd) and NACs. Their rules have to be locally confluent, so that the polynomial naive LGG parsing algorithm in [11] works. Backtracking for handling recognized critical rule pairs is not supported.

In Section 2 we briefly review the GenGEd environment for the visual specification of VLs. The parsing facilities and their usage in GenGEd are proposed in Section 3, and illustrated by a small example (a subset of the well-known UML class diagrams). Although this example is not very expressive, it is suitable for illustrating the concepts. In Section 4 we conclude.

2 The GenGEd Environment

The GenGEd environment implements concepts for the visual specification of VLs [1]. A VL-specification is given by a visual alphabet and a visual grammar. In the visual alphabet the types of symbols and links occurring in a VL are specified. The visual grammar consists of a start expression and a set of context-sensitive grammar rules. Originally the grammar rules define the syntax-directed editing commands of a language-specific graphical editor, i.e., the visual grammar does not only comprise language-generating rules but a convenient set of editing rules as well. In the following we show that the concepts of VL-specifications can be easily extended by the specification of parsing.

GenGEd is based on algebraic graph transformation [4] and graphical constraint solving [7]. A visual alphabet is represented by an attributed graph structure signature [2] and a constraint satisfaction problem defining positions and sizes of visual elements. Correspondingly, a visual grammar is represented by an attributed graph structure grammar where the constraint satisfaction problem of each visual expression is satisfied. Moreover, we distinguish two syntactical levels, namely the *abstract syntax* describing the logical part of a VL, and the *concrete syntax* denoting the layout.

According to the constituents of a VL-specification, the GenGEd environment comprises an alphabet editor and a grammar editor. The specified alphabet is the input of the grammar editor, where so-called *alphabet rules* are generated defining the editing commands of this editor. In this way it is guaranteed that only correct visual expressions can be defined by a language designer. For the transformation of visual expressions according to the abstract syntax the Agg system [6] is used. The graphical constraints are solved by the constraint solver ParCon [7].

[2] A graph structure signature is an algebraic signature according to [5] with unary operation symbols.

The parsing algorithm proposed in [2] (which is based on Contextual Layered Graph Grammars (*CLGG*) and critical pair analysis) is now implemented using the Agg system, hence we call it Agg *graph parser*. In GenGEd we integrated the Agg graph parser such that we yield a parser for visual languages. In this sense, not only syntax-directed editing but also free editing is available in specific graphical editors generated by GenGEd, similar to [8]. As before, the alphabet editor supports the definition of visual alphabets comprising the types for symbols and links. Based on a visual alphabet, the grammar editor may be used in two ways: for the definition of comprehensive syntax-directed editing rules as well as for the definition of a parse specification. The latter one is explained in the following.

3 Parsing of Class Diagrams

As already mentioned before, a visual alphabet describes the types of symbols (vertices) and links (edges) of a VL. Figure 1 illustrates the visual alphabet for a subset of UML class diagrams. It comprises the symbols needed and explains how these symbols are linked. In the top the abstract syntax is shown where the lexical symbols Package, Class and Assoc (association) are framed by rectangles, and the attribute symbol CN (class name) of type String by rounded rectangles. The arrows indicate the links between the symbols. The symbol's layouts are connected with the abstract syntax by so-called *layout operations* illustrated by dashed arrows. The constraints which have to be defined for each (abstract) link are illustrated by dotted arrows between the symbol layouts.

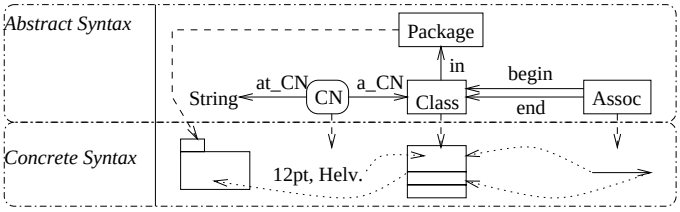

Fig. 1. Visual alphabet of the class diagram language.

A visual alphabet is the basis to define a parse specification using the GenGEd grammar editor. Based on the Agg graph parser, a parse specification consists of a parse grammar (which can be defined using the means of the VL), a layer function, and critical pairs.

Graph rules occurring in a parse grammar consist of a left-hand side (L) and a right-hand side (R) over typed (labeled) graphs. Parts of both rule sides are related to each others. The related parts are preserved during a graph transformation. All non-related graph objects of L are deleted, all non-related objects of R are created. Moreover, a rule may contain a set of NACs specifying exactly those fractions of matching situations that *must not* exist for a rule to be applicable.

Assigning rules as well as vertex and edge types to layers such that a certain layering condition is satisfied (cf. [2]), the layer-wise application of rules (according to the *rule layer*) to a given terminal graph always terminates. Roughly speaking, the layering condition is fulfilled if each rule deletes at least one vertex or edge coming from a lower level (*deletion layer*) and creates graph objects of a higher level (*creation layer*).

Critical pair analysis [10,9,12] can be used to make parsing by graph transformation more efficient: decisions between conflicting rule applications are delayed as far as possible. This means to apply non-conflicting rules first and to reduce the graph as much as possible. Afterwards, rule application conflicts are handled by creating decision points for the backtracking part of the parsing algorithm. For critical pair analysis of CLGG rules, a layer-wise analysis is sufficient, since a rule of an upper layer is not applied as long as rules of lower layers are still applicable.

In the GenGEd grammar editor, the critical pairs are generated automatically from the Agg graph parser, but the remaining constituents the language designer has to define. For our example of simplified UML class diagrams, the parse rules express the deletion of visual symbols such that for each lexical symbol of the visual alphabet there is one parse rule. These rules and the layer function are proposed in the following.

Figure 2 (a) illustrates the parse rule for packages. Packages can be deleted if they are empty. If the package where the rule should be applied to, is not empty, the dangling condition prohibits the application of this rule. The rule allowing for the deletion of association symbols is illustrated by Figure 2 (b). Here we consider only classes of the same package to be related by association symbols.

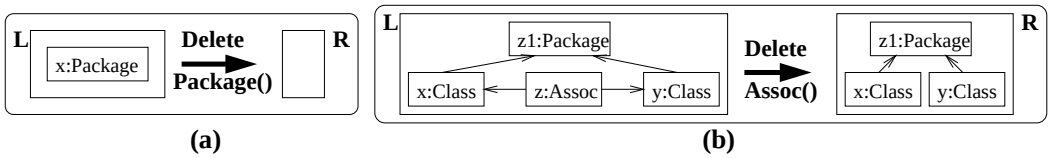

Fig. 2. Parse rule for package symbols (a) and for association symbols (b).

The rule supporting the deletion of class symbols is shown in Figure 3. According to the visual alphabet, a class symbol always has to be linked to a package which is expressed by the left-hand side L of the rule. Moreover, we expect that the user always inserts a class symbol together with a class name represented by the node x':CN holding the value represented by the variable cn. The NAC states that the class name has to be unique in one package.

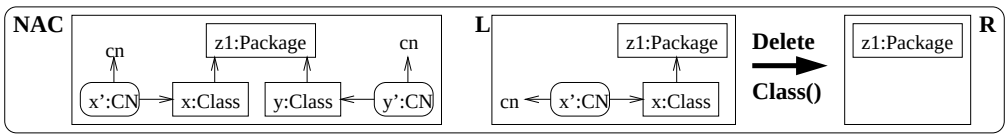

Fig. 3. Parse rule for class symbols and class names.

The layer function for our small example is given below. Thereby we use the abbreviations *dl* for *deletion layer*, *cl* for *creation layer*, and *rl* for *rule layer*. Note that the rule layer supports the ordering of rule application, whereas the deletion and the creation layer are necessary for the termination of the parsing algorithm. Note that the language designer must not define creation nor deletion layers for links; those are generated automatically in dependence of the symbols the links have in their domain, i.e., the source vertices of the corresponding link edges.

Type dependent layers					**Rule layer**		
dl(Assoc)	=	cl(Assoc)	=	0	rl(DeleteAssoc())	=	0
dl(Class)	=	cl(Class)	=	1	rl(DeleteClass())	=	1
dl(Package)	=	cl(Package)	=	2	rl(DeletePackage())	=	2

For the critical pair analysis which must be done only once, the Agg graph parser is called with the parse grammar and the layer function. The resulting parse specification and the visual alphabet is the input of the graphical editor where the user can manipulate visual expressions (diagrams) in a free editing style. In order to check the visual expression against the visual syntax, the Agg graph parser gets the parse specification together with the visual expression as input and checks whether the expression is correct or not. The result will be illustrated in the graphical editor.

4 Conclusion

Graph parsing based on critical pair analysis can be used for efficient VL parsing. The VL parser integrated in GenGEd is based on a new graph parsing component of Agg. First experiences on free editing of class diagrams have been made. We are going to consider more sophisticated VLs in the future as, e.g. Statecharts. Furthermore, the implemented parsing algorithm has to be compared with related approaches concerning efficiency.

References

[1] R. Bardohl. GenGEd – *Visual Definition of Visual Languages based on Algebraic Graph Transformation.* PhD thesis, TU Berlin, Germany, 1999. Verlag Dr. Kovac, 2000.

[2] 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.

[3] G. Costagliola, A.D. Lucia, S. Orefice, and G. Totora. Positional Grammars: A Formalism for LR-Like Parsing of Visual Languages. In K. Marriott and B. Meyer, editors, *Visual Language Theory*, pages 171–192, Springer, 1998.

[4] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and Corradini. Algebraic Approaches to Graph Transformation II: Single Pushout Approach and Comparison with Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, pages 247–312. World Scientific, 1997.

[5] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specifications 1: Equations and Initial Semantics*, volume 6 of *EACTS Monographs on Theoretical Computer Science*. Springer, Berlin, 1985.

[6] C. Ermel, M. Rudolf, and G. Taentzer. The AGG-Approach: Language and Tool Environment. In H. Ehrig, G. Engels, J.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, pages 551–604. World Scientific, 1999.

[7] P. Griebel. *Paralleles Lösen von grafischen Constraints.* PhD thesis, University of Paderborn, Germany, February 1996.

[8] O. Köth and M. Minas. Generating Diagram Editors Providing Free-Hand Editing as well as Syntax-Directed Editing. In *Proc. GRATRA'2000 - Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems*, pages 32–39. Technische Universität Berlin, Germany, March 2000.

[9] M. Löwe and J. Müller. Critical Pair Analysis in Single-Pushout Graph Rewriting. In G. Valiente Feruglio and F. Rosello Llompart, editors, *Proc. Colloquium on Graph Transformation and its Application in Computer Science*. Technical Report B-19, Universitat de les Illes Balears, 1995.

[10] Detlef Plump. Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In M.R Sleep, M.J. Plasmeijer, and M. C.J.D. van Eekelen, editors, *Term Graph Rewriting*, pages 201–214. Wiley, 1993.

[11] J. Rekers and A. Schürr. Defining and Parsing Visual Languages with layered Graph Grammars. *Journal of Visual Languages and Computing*, 8(1):27–55, 1997.

[12] T. Schultzke. Concepts and Implementation of a Parser for Visual Languages. Master's thesis (in German), TU Berlin, Germany, 2001.

[13] D.-Q. Zhang and K. Zhang. Reserved graph grammar: A specification tool for diagrammatic VPLs. In *Proc. IEEE Symposium on Visual Languages (VL'97)*, pages 288–295. IEEE Computer Society Press, 1997.