

# Auto-completion for Diagram Editors based on Graph Grammars

Steffen Mazanek, Sonja Maier, Mark Minas  
Universität der Bundeswehr München, Germany  
{steffen.mazanek, sonja.maier, mark.minas}@unibw.de

## Abstract

Graphs are known to be well-suited as an intermediate data structure in diagram editors. The syntax of a particular visual language can be defined by means of a graph grammar. In recent work we have proposed approaches to graph completion: given a possibly “incomplete” graph, this graph is modified in such a way that the resulting graph is a member of the grammar’s language. In this paper we describe how graph completion can be used to realize diagram completion, an important requirement for the realization of content assist in diagram editors. With our approach, the advantages of free-hand and structured editing can be effectively combined: drawing of diagrams with maximal freedom and powerful guidance whenever needed.

## 1 Introduction

Nowadays, visual domain-specific languages are very popular. Many ways of simplifying the development of powerful and feature-rich diagram editors have been proposed. Still one feature of textual IDEs has not made its way to diagram editors yet: the syntax-aware, context-sensitive editing assistance we are so fond of.

Consider some typical user Joe. Normally, Joe has to decide between two kinds of diagram editors: A structure editor offers him operations that transform correct diagrams into (other) correct diagrams. Joe really likes this kind of guidance, because that makes editing much easier. But he also likes to draw his diagrams freely just following the flow of his uninhibited associations. A free-hand editor allows him to arrange diagram components on the screen without any restrictions. Using syntax analysis, the free-hand editor decides whether the drawing conforms to the visual language and what structure Joe intended to define.

Following the approach presented in this paper, Joe does not have to decide in advance between the two alternatives

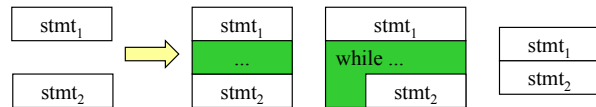


Figure 1. Completions of a broken NSD

anymore. Rather, he can get the best of both worlds: He can draw his diagram with maximal freedom, and additionally gets assistance by the editor whenever needed. Initially, this assistance helps Joe to learn the particular language. As he gets better, he will further benefit from the powerful completion engine at his fingertips, just like expert users benefit from the assistance provided by modern IDEs.

Consider Joe, how he starts an editor for Nassi-Shneiderman diagrams (NSD) and quickly draws two simple statements (left-hand side of Fig. 1). Unfortunately, his diagram is incorrect. Luckily, he knows how to ask the editor for assistance. He gets several suggestions, three of which are shown on the right-hand side of the figure. So Joe learns that he can complete his simple diagram in a variety of ways, e.g., by inserting an additional component.

In this paper, we propose a method for diagram completion that is applicable to visual editors based on graph grammars. Since the graph parser can additionally compute so-called graph completions, most of the required information is readily available, i.e. only little extra specification is needed. We also outline how this method is actually implemented within the DIAGEN editor generator framework.

## 2 Graph Grammars and Graph Completions

In, e.g., the DIAGEN system [7], (hyper-)graphs have proved to be suitable models for diagrams: Visual components are mapped to hyperedges, an extension of normal graph edges that connect an arbitrary number of nodes. Nodes represent a component’s attachment points which help to model connections to other components.

A typical Nassi-Shneiderman diagram and its corre-

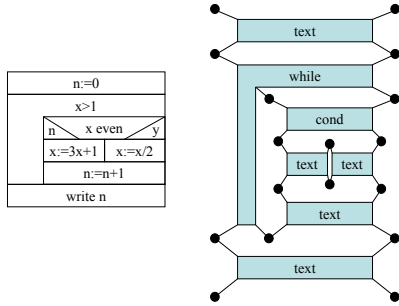


Figure 2. A typical NSD and its graph model

sponding graph model are shown in Fig. 2. (Hyper-)edges are represented by boxes marked with labels. The black dots represent nodes. A line between an edge and a node indicates that the node is connected to that edge.

In DIAGEN the syntax of visual languages is defined using (a slightly extended version of) hyperedge replacement grammars HRG [2]. This graph grammar formalism is quite similar to context-free string grammars. The productions of an HRG describing the language of NSDs are given in Fig. 3. We use the symbol  $::=$  to keep the left-hand side of productions apart from their corresponding right-hand sides; vertical bars separate right-hand sides of productions with the same left-hand side. Nonterminal edge labels are “NSD” (that is also the start label) and “Stmt”, terminal ones are “text”, “while” and “cond”. Node labels are used to identify corresponding nodes of both sides of a production.

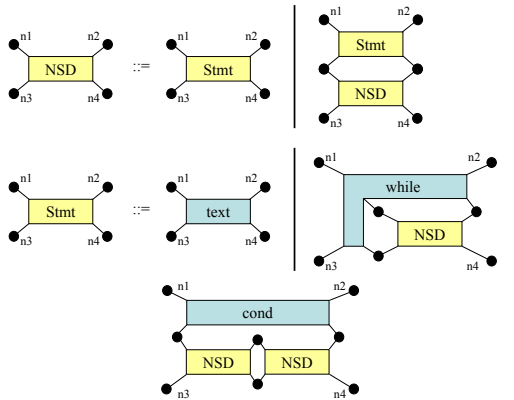


Figure 3. HRG of NSDs

Rewriting is done by replacing a nonterminal hyperedge of a given hypergraph with the hypergraph given in the right-hand side of a corresponding production. This graph has to be glued to the remaining graph by fusing corresponding nodes. The language defined by an HRG consists of all graphs that can be derived in an arbitrary number of steps from the start symbol and whose edges are labeled with terminal labels only. Fig. 4 shows a short derivation.

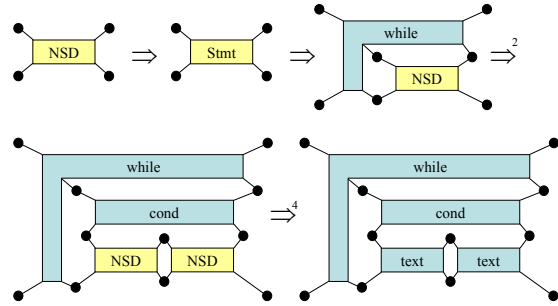


Figure 4. A possible derivation sequence

### Graph Completions

Given a graph  $H$  and an HRG  $G$ . In this paper, a completion of  $H$  wrt  $G$  is a graph  $H'$  that belongs to the language of  $G$  and that results from  $H$  by successively performing one of the following kinds of modifications, both illustrated in Fig. 5:

- Insertion of new edges into  $H$  as in  $C_1$  and  $C_2$ .
- Identification of distinct nodes of  $H$  as in  $C_3$  and  $C_4$  (denoted by  $\sim$  in the figure).

The particular number of edges inserted into  $H$  is called the size of the completion. Note that we do not prevent the input graph  $H$  from already belonging to the language of  $G$ .

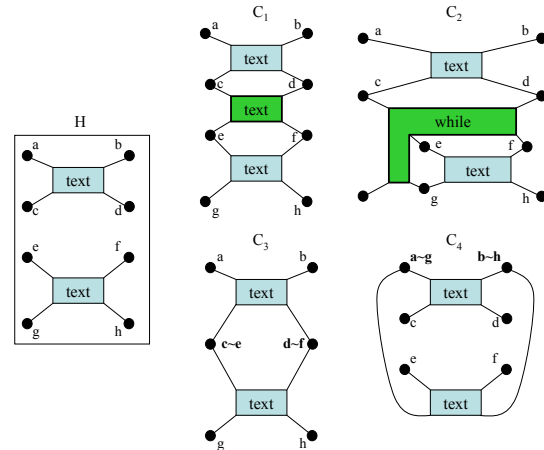


Figure 5. Completions on the graph level

In previous work we have introduced two different approaches to graph completion wrt hyperedge replacement grammars that both can be used as a back-end for diagram completion: In [6] we have proposed the combinator approach to graph parsing. Due to their logic nature, the resulting graph parsers can also be used as language generators or for the completion of partial graphs. The basic idea

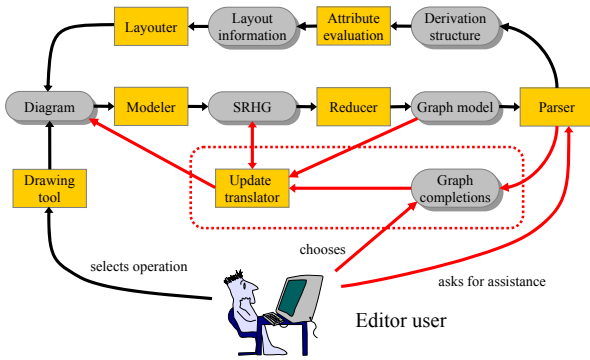


Figure 6. Extended DIAGEN editing process

has been to feed logical variables into the graph. These variables are instantiated afterwards as a side effect of parsing. This approach is straightforward, but not very efficient.

Therefore, we have suggested an alternative approach to graph parsing with completion [5]. We exploit dynamic programming techniques similar to the algorithm of Cocke, Younger and Kasami. By this means completions can be computed in a reasonably efficient way. However, in both approaches the size of the resulting completions has to be restricted in advance by an additional input parameter.

### 3 From Graphs to Diagrams

In this section we describe how we translate graph completions into diagram completions. A bird's eye view on our editing process is shown in Fig. 6. In fact, it is an extension of the normal editing process for DIAGEN editors [7] that we briefly introduce first.

The user of a DIAGEN editor can freely edit diagrams using the drawing tool. After each editing operation a chain of processing steps is performed. First, the modeler derives the so-called spatial relationship hypergraph (SRHG) from the arrangement of diagram components. Thereby, spatial relationship edges are introduced where components are connected in a way relevant for the particular visual language. An SRHG of an example NSD is part of Fig. 7.

Thereafter, a reduction step is performed to reduce complexity very similar to lexical analysis in string parsing. This allows for more efficient parsing and readable grammars. In case of NSDs, nodes connected via spatial relationship edges are unified, resulting in a graph model like the one shown in Fig. 2.

Next, the parser performs the syntactical analysis of the graph model. Thereby, a derivation structure is constructed (if any). In the following, this derivation structure can be used by the layouter and for highlighting correctly recognized parts of the diagram.

We have extended this process with diagram comple-

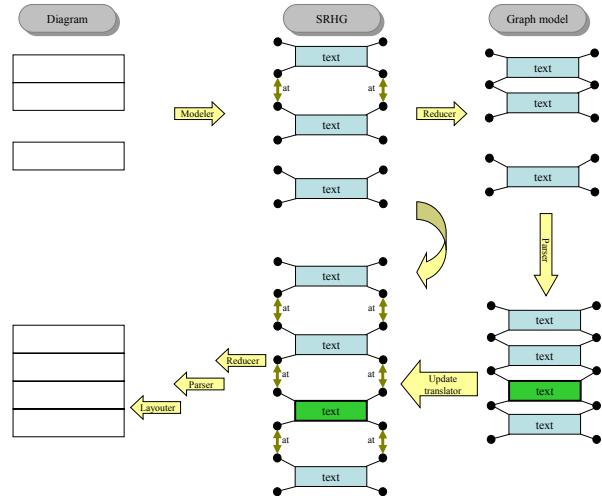


Figure 7. Processing steps by example

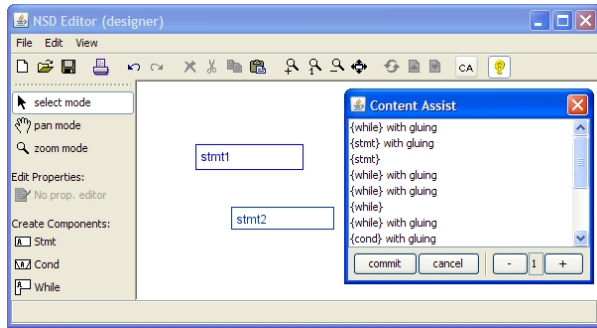
tion in the following way: If the user explicitly asks for assistance, the parser is triggered with the desired size of completions as a parameter. It computes all possible graph completions up to this size (cf. [5]). From those, the user has to choose. Next, the selected completion is embedded into the SRHG using the update translator. This procedure is language-specific and has to be provided by the editor developer. It has to handle both kinds of modifications a completion might consist of, i.e., embedding of new edges and gluing of existing nodes. Then we are practically done: We can simply reduce and parse again and the layouter will take care of arranging the new components within the actual diagram.

A common way to specify the layout for DIAGEN editors is by using constraints. Therefore, as suggested in [8], the productions of the grammar are enriched with layout constraints. The derivation then establishes a constraint satisfaction problem whose solution is a layout for the diagram. We currently rely on the constraint solver QOCA [4] in order to get solutions of minimal changes.

Constraints generally are known to be well-suited for the description of layout on a rather high level of abstraction. However, in our domain an additional benefit becomes noticeable: The constraints normally used for diagram beautification can be directly used to embed the completions, i.e., no extra specification effort is necessary for this purpose.

### 4 The User Interface

A convenient user interface is crucially important for a new technique to be accepted by users. Therefore, we have started to experiment with different ways of accessing the completions. Here, the main problem is that computing the new diagram from the abstract completion is a quite expen-



**Figure 8. NSD editor and assistance dialog**

sive operation, in particular due to layouting. Furthermore, practical diagrams can be quite large and there can be many different completions, so that a preview of all possible solutions is impossible in general. Zooming only to the changed parts might also not always be feasible.

Therefore, at the moment the UI works as follows: If the user asks for assistance, a dialog is opened as shown in Fig. 8. The number in the lower right corner indicates the size of the completions currently displayed. This parameter can be changed by pushing the corresponding buttons. For the example diagram there are quite a few completions of size one, some of which also glue nodes. If a user clicks on an item in the list, this particular completion is shown as a preview. That way several completions can be tried out and inspected by the user. Only after a commit the selected change is actually applied to the diagram.

## 5 Related Work

Not much research has been done wrt diagram completion yet. GMF [3] editors assist the user with action toolbars and special connector handles. In [9] an approach for completion in the context of metamodel-based editors has been proposed, which employs constraint-logic programming. The grammar-based system VLDESK [1] provides support for so-called symbol prompting. Here, the parsing table is exploited to extract information about possible contexts of a particular symbol. In TIGER [10] complex editing commands can be specified by means of graph transformation rules. While this is a powerful way to specify editing operations, it is also quite tedious and error-prone. With our approach a set of applicable commands, which automatically preserve or even establish the syntactical correctness of the resulting diagram, is computed on the fly.

## 6 Conclusion and Future Work

The main contribution of this paper is an approach to diagram completion for diagram editors based on graph gram-

mars. We use the parser to gather the information how a given diagram can be completed on the abstract syntax level. Thereafter, an update translator component embeds the completion into the original diagram and the layout engine places the new components at proper positions.

Our approach supports both inexperienced users and experts. Beginners can easily explore the language at hand. Experts can draw complex diagrams much faster thanks to arbitrary switches between free-hand and structured editing.

So far, we only have realized editors for NSDs, flowcharts and trees. In future we will specify more editors to demonstrate the broad applicability of our approach. Also, we need to investigate appropriate user interaction mechanisms in greater depth. One issue is, that small changes on the abstract syntax level may result in dramatic changes of the final diagram. Since completions causing minimal diagram changes are generally preferred, we need to develop a cost model that comprises information from both the abstract and the concrete syntax level.

## References

- [1] G. Costagliola, V. Deufemia, G. Polese, and M. Risi. Building syntax-aware editors for visual languages. *Journal of Visual Languages and Computing*, 16(6):508–540, 2005.
- [2] F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 2, pages 95–162. World Scientific, 1997.
- [3] R. Gronback et al. Eclipse Graphical Modeling Framework, 2008. [www.eclipse.org/gmf](http://www.eclipse.org/gmf).
- [4] K. Marriott and S. S. Chok. Qoca: A constraint solving toolkit for interactive graphical applications. *Constraints*, 7(3-4):229–254, 2002.
- [5] S. Mazanek, S. Maier, and M. Minas. An algorithm for hypergraph completion according to hyperedge replacement grammars. In *Proc. of the 4th Intl. Conference on Graph Transformation*, LNCS. Springer, 2008.
- [6] S. Mazanek and M. Minas. Functional-logic graph parser combinators. In *Proc. of the 19th Intl. Conference on Rewriting Techniques and Applications*, LNCS. Springer, 2008.
- [7] M. Minas. Concepts and realization of a diagram editor generator based on hypergraph transformation. *Science of Computer Programming*, 44(2):157–180, 2002.
- [8] M. Minas and G. Viehstaedt. Specification of diagram editors providing layout adjustment with minimal change. *Proc. 1993 IEEE Symposium on Visual Languages*, pages 324–329, 1993.
- [9] S. Sen, B. Baudry, and H. Vangheluwe. Domain-specific model editors with model completion. In *Multi-paradigm Modelling Workshop at MoDELS 2007*, 2007.
- [10] G. Taentzer, A. Crema, R. Schmutzler, and C. Ermel. Generating domain-specific model editors with complex editing commands. In *Proc. Third Intl. Workshop and Symposium on Applications of Graph Transformation with Industrial Relevance*, 2007.