Generating Diagram Editors Providing Free-Hand Editing as well as Syntax-Directed Editing

Oliver Köth and Mark Minas

Lehrstuhl für Programmiersprachen
Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
minas@informatik.uni-erlangen.de

Abstract. Diagram editors which are tailored to a specific diagram language typically support either syntax-directed editing or free-hand editing, i.e., the user is either restricted to a collection of predefined editing operations, or he is not restricted at all, but misses the convenience of such complex editing operations. This paper proposes a concept for incorporating both modes into one editor in order to get the combined advantages. This proposal extends work on free-hand editors which are generated from a formal hypergraph grammar specification by the diagram editor generator *DiaGen*.

1 Introduction

Diagram editors are graphical editors which are tailored to a specific diagram language; they can be distinguished from pure drawing tools by the capability of "understanding" edited diagrams to some extent. Current diagram editors support either *syntax-directed editing* or *free-hand editing*.

Syntax-directed editors provide a set of editing operations. Each of these operations is geared to modify the meaning of the diagram. This editing mode requires an *internal diagram model* that is primarily modified by the operations; diagrams are then updated according to their modified model. These models are most commonly described by some kind of graph; editing operations are then represented by graph transformations (e.g. [2, 5]).

Diagram editors providing free-hand editing are low-level graphics editors which allow the user to directly manipulate the diagram. The graphics editor becomes a diagram editor by offering only pictorial objects which are used by the visual language and by combining it with a parser. A parser is necessary for checking the correctness of diagrams and analyzing as well as recognizing the syntactic structure of the diagram. There are grammar formalisms and parsers that do not require an internal diagram model as an intermediate diagram representation, but operate directly on the diagram (e.g., constraint multiset grammars [3]). Other approaches use an internal model which is analyzed by the parser. Again, graphs are the most common means for describing such a model.

The advantage of free-hand editing over syntax-directed editing is that a diagram language can be defined by a concise (graph) grammar only; editing operations can be omitted. The editor does not force the user to edit diagrams in a certain way since there is no restriction to predefined editing operations. However, this may turn out to be a disadvantage since editors permit to create any diagram; they do not offer explicit guidance to the user. Furthermore, free-hand editing requires a parser and is thus restricted to diagrams and (graph) grammars which offer efficient parsers.

So far, diagram editors either support syntax-directed editing or free-hand editing. An editor that supports both editing modes at the same time would combine the positive aspects of both editing modes and reduce their negative ones. Despite this observation, there is only one such approach known to us: Rekers and Schürr propose to use two kinds of graphs as internal representations of diagrams [10]: the *spatial relationship graph* (SRG) abstracts from the physical diagram layout and represents higher level spatial relations. Additionally, an *abstract syntax graph* (ASG) that represents the logical structure of the diagram is kept up-to-date with the SRG. Context-sensitive graph grammars are used to define the syntax of both graphs. Free editing of diagrams is planned to modify the first graph, syntax-directed editing is going to modify the second. In each case, the other graph is modified accordingly. Therefore, a kind of diagram semantics is available by the ASG. However, this approach requires almost a one-to-one relationship between SRG and ASG. This is not required in the approach of this paper. We will come back to this approach in the conclusions (cf. Sect. 6).

This paper describes another, but related, concept for automatically generating diagram editors that support both editing modes at the same time. This concept extends *DiaGen*, the *Dia*gram editor *Gen*erator for pure free-hand editors, which has been described in [7]: It uses an internal hypergraph model which is parsed according to some hypergraph grammar in order to analyze the syntactic structure of the hypergraph (and thus of the diagram). Attribute evaluation based on this structure is then used for creating a user-specified semantic representation of the diagram. This paper makes two new contributions: it describes how such a free-hand editor is extended by an additional syntax-directed editing mode, which also requires an automatic layout mechanism for diagrams. Such a constraint-based layout mechanism which is used for both syntax-directed editing and free-hand editing is briefly outlined, too.

The next section briefly introduces sequential function charts as the running example of this paper. Section 3 outlines our existing work on free-hand diagram editors, which is extended by syntax-directed editing in Sec. 4. An automatic layout mechanism, which is required by syntax-directed editing, is outlined in Sec. 5. Section 6 concludes the paper.

2 Example: Sequential Function Charts

Throughout this paper, we will use sequential function charts (SFC) as running example. *SFC* is a visual programming language for Programmable Logic Con-

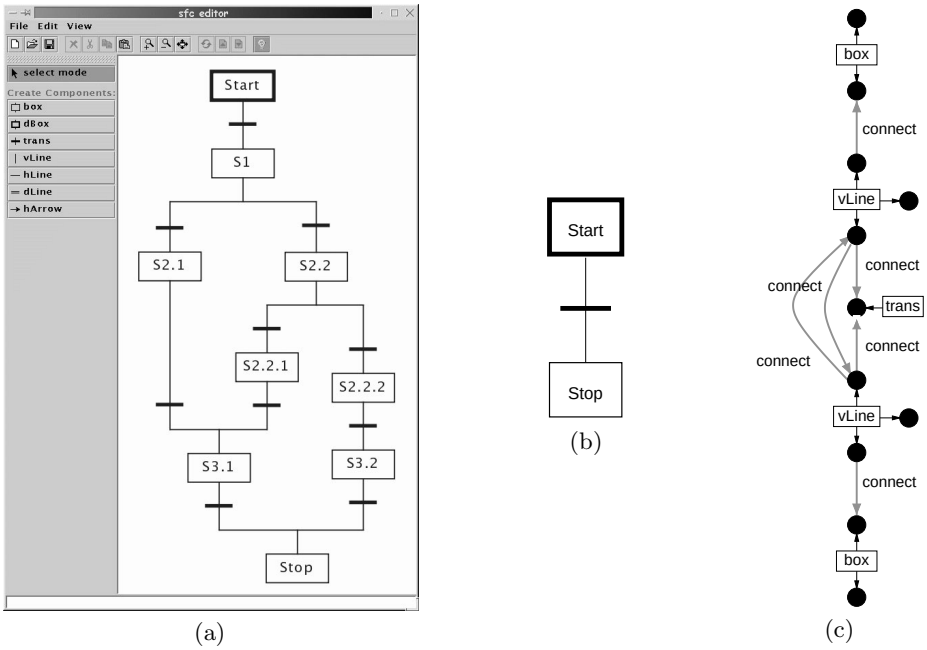

Fig. 1. Two SFC diagrams (a) and (b) – the first one shown as screenshot of the generated SFC editor – and the spatial relationship hypergraph (c) of the diagram in (b).

trollers (PLCs), which are widespread devices for controlling machines in nearly every application domain. SFC as a language has become part of the IEC 1131 standard [1] and is quite similar to Petri nets: an SFC mainly consists of *steps* and *transitions* which build a bipartite graph: Steps may be connected to transitions and transitions to steps. A step becomes active if its predecessor transition fires, and it becomes inactive again if its successor transition fires. Steps of SFC represent a certain part of a PLC program which is active if and only if the step is active. Transitions carry boolean conditions which define when a transition fires. SFC diagrams are thus used to structure a PLC program on a very high level.

Figure 1a and b show two sample SFC diagrams. Please note that transitions and steps are connected by vertical lines from top to bottom. Alternatives are represented by plain horizontal lines, the initial step by a fat borderline. SFC also offers additional language constructs which are omitted here.

3 Free-Hand Editing

This section describes the idea of free-hand editing as it has been presented in [7]. Free-hand editors for a specific diagram language are generated from a specification which describes the diagram language and the translation process

from the diagram to some user-specified semantic representation. The editor performs a sequence of three steps after each editing operation: the scanning, the reduction, and the parsing step.

Scanning step: Diagram components (e.g., boxes and vertical lines in SFC) have *attachment areas*, i.e., the parts of the components that are allowed to connect to other components (e.g., top and bottom of a box, which are connected to vertical lines). The most general and yet simple formal description of such a component is a hyperedge which connects to the nodes which represent the *attachment areas*[1] of the diagram components. These nodes and hyperedges first make up an unconnected hypergraph. The *scanner* connects nodes by additional edges if the corresponding attachment areas are related in a specified way, which is described in the specification. The result of this scanning step is the *spatial relationship hypergraph* (SRHG) of the diagram. Figure 1c shows the SRHG of the SFC diagram shown in Fig. 1b. Nodes are represented by black dots, hyperedges either by gray arrows (relationships between attachment areas) or by rectangles (diagram components) that are connected to their nodes ("attachment areas") by small arrows.

Reduction step: SRHGs tend to be quite large even for small diagrams (see Fig. 1c). In order to allow for efficient parsing, a reduced *hypergraph model* (HGM) is created from the SRHG first. The reducer is specified by some transformations that identify those sub-hypergraphs of the SRHG which carry the information of the diagram and build the HGM accordingly. This step is similar to the lexical analysis step of traditional compilers.

Parsing step: The syntax of the hypergraph models of the diagram language—and thus the syntax of the language—is defined by a hypergraph grammar. *DiaGen* supports context-free hypergraph grammars with embeddings, i.e., productions are either context-free ones with a single nonterminal hyperedge on the LHS of the production, or they are embeddings where the RHS of the production is the same as its LHS, but with an additional hyperedge.

4 Syntax-directed Editing

As discussed in the introduction, syntax-directed editing has several important benefits. Other approaches for free-hand editing which do not make use of abstract internal models (e.g., the Penguins system based on constraint multiset grammars [3, 4]) cannot extend free-hand editing by syntax-directed editing, which requires such an abstract model. But since our approach is based on such a model (the SRHG), it is quite obvious to offer syntax-directed editing, too. However, free-hand editing using a parser requires that the hypergraph grammar remains the only syntax description of the HGM and thus the diagram language. Syntax-directed editing operations must not change the syntax of the

[1] Plain edges instead of hyperedges are in general not sufficient: E.g., in SFC, lines can connect to other lines at their end points or with some "inner point". Those situations have to be well distinguished in SFC.

diagram language; they can only offer some additional support to the user. This requirement has two immediate consequences:

(1) It is possible to specify editing rules that deliberately transform a correct diagram into an incorrect one with respect to the hypergraph grammar. This might appear to be an undesired feature; but consider the process of creating a complex diagram: the intermediate "drawings" need not, and generally do not make up a correct diagram, only the final "drawing". In order to support those intermediate incorrect results, syntax-directed editing operations have to allow for such "disimprovements", too.

(2) Editing operations are quite similar to *macros* in off-the-shelf text and graphics editors; they combine several actions, which can also be performed by free-hand editing, into one complex editing operation. However, syntax-directed editing rules are actually much more powerful than such macros which offer only recording of editing operations and their playback as a complex operation: syntax-directed editing operations also take care of providing a valid diagram layout where this is possible (incorrect diagrams have no valid layout in general.) Furthermore, editing operations can take into account context information, and they may have rather complex application conditions.

This makes use of graph transformation an obvious choice for adding syntax-directed editing to the free-hand editing mode: Editing operations are specified by hypergraph transformations on the SRHG. Whenever the SRHG has been changed by some transformation, it has to be (re)parsed. The results of the parser are then used to indicate correct (sub)diagrams and to create a valid layout for them (see Sec. 5). Please note that the SRHG is directly modified by the transformation rules; the scanning step, which is necessary for free-hand editing, does not take place.

Figure 2 shows an example of a syntax-directed editing operation for SFC: a part of a chart, which is identified by its top-most step and its last transition (indicated by the dark gray background in Fig. 2), is inserted between a transition and a step. This editing operation is easily specified by a hypergraph transformation rule on the SRHG of the diagram as shown in Fig. 2: Corresponding nodes of the LHS and RHS have the same labels, the dark gray lines between the SFC situation and the LHS of the rule indicate which hyperedge of the LHS corresponds to which part of the diagram. The parts of the LHS and RHS of the rule which are contained in an "?"-marked oval are optional parts which might be missing. When the rule is applied, the connection between the transition and the step at the target position is removed (the *connect*-edge from node c to d), the sub-chart that is going to be moved to its new position is cut off from its context (the optional *connect*-edges between nodes f and g resp. j and i are removed), and then inserted into its new context (a line-component together with its hyperedge is added at nodes m, n, and o along with the new *connect*-edges). Please note that the application of this rule does not yet alter the position of any diagram component. This is the task of the automatic layout which takes place after the resulting SRHG has been analyzed syntactically.

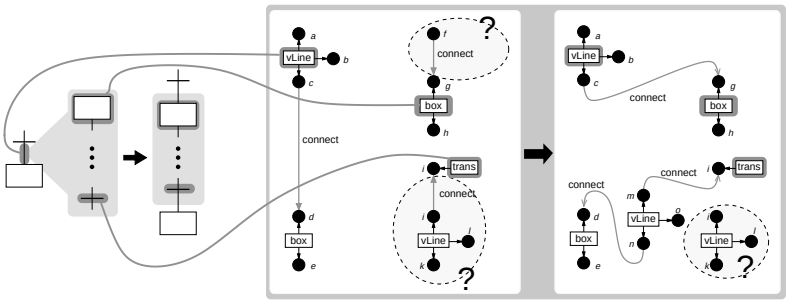

Fig. 2. A syntax-directed editing operation for SFC: A sub-chart is inserted between a transition and a step which is represented by a transformation on the SRHG.

In *DiaGen*, syntax directed editing operations are specified in terms of simple *rules* and complex *operations* quite similar to rules and transformation units in GRACE [6]. Rules are specified as pairs of LHS and RHS with optional application conditions which may consist of negative context or path-expressions which have to be satisfied when a rule is going to be applied. Operations are defined in terms of rules[2]; control conditions describe valid sequences of rule applications. Each syntax-directed editing operation is specified by such a complex operation.

An important issue of syntax-directed editing is the question how to select those parts of the diagram that are affected by the application of an editing operation. In *DiaGen*, this has been solved by adding parameters to complex operations. When the user selects an editing operation for application, the editor requests that the user has to specify a single diagram component for each of the parameters of the operation. The hyperedges that internally represent these components specify a partial match which is then used to select where the operation and its rules have to be applied.

5 Automatic Layout

Transformations on the SRHG modify the structure of the internal model, but they do not describe their effects on the position or the size of the diagram components; an automatic layout mechanism is needed. Since a correct layout also reflects the syntax of a diagram, it is quite obvious to make use of the syntactic structure as it is analyzed by the reduction step and the parsing step. For this purpose, we can refer to previous work of ours on automatic layout based on the syntactic structure of the diagram [9]: The main idea is to describe a diagram layout in terms of values which are assigned to the attributes of the diagram components (e.g., their position). A valid diagram layout is specified by a set of constraints on these attributes; the constraint set is determined by the syntactic structure of the diagram. This works as follows: Hyperedges of the SRHG and terminal as well as nonterminal hyperedges of the HGM carry

[2] The current implementation does not allow to "invoke" operations from an operation.

attributes, reduction step rules and grammar productions are augmented by constraints on their accessible attributes. These constraints are added to the set of constraints which specify a diagram layout whenever the corresponding rule or production is instantiated during the reduction step or parsing process.

It is important to define layout constraints not only in the hypergraph grammar which is used during the parsing step, but also in the rule set which specifies the reduction step. This is so because the reduction step may "reduce away" the explicit representation of some specific diagram components (e.g., lines in our SFC example). If we had restricted specification of layout constraints to the hypergraph grammar, we would not be able to describe the layout of those diagram components.

This layout mechanism is not restricted to syntax-directed editing. The same information is also available during free-hand editing. Editors specified and generated by *DiaGen* therefore offer an *intelligent diagram* mode where diagram components may be modified arbitrarily, but the other components, especially their position, may be affected by these modifications, too. The constraints take care of modifying the overall appearance of the diagram such that its syntax is preserved and the layout beautified. This work on *intelligent diagrams* is similar to the approach by Chok, Marriott, and Paton [4].

6 Conclusions

This paper has outlined work on graphical diagram editors that support free-hand editing as well as syntax-directed editing. By supporting both editing modes in one editor, we hope to combine the positive aspects of both modes, i.e, unrestricted editing capabilities and convenient syntax-directed editing.

The approach which has been presented in this paper appears to be quite similar to the approach of Rekers and Schürr [10] which has already been outlined in Sec. 1. Both approaches make use of two hypergraphs resp. graphs. The spatial relationship graph (SRG) in Reker's and Schürr's approach is quite similar to our SRHG. But their abstract syntax graph (ASG) has been introduced for a different reason than our HGM: SRHGs (and also SRGs) are generally quite complicated such that there is no hypergraph parser which can analyze an SRHG. Therefore, we preprocess the SRHG and parse the much simpler HGM instead of the SRHG. As we have demonstrated, parsing of the HGM can be performed efficiently. However in Reker's and Schürr's approach, SRG and ASG are always strongly coupled since they use *triple graph grammars* for defining the syntax of the SRG and the ASG with one formalism; the ASG has not been introduced for reducing complexity. Instead, a graph grammar parser has to analyze the SRG directly; the ASG is not parsed, it is created as a "side-effect" of the parsing of the SRG during free-hand editing. The requirement for a graph parser for the SRG imposes a strong restriction on this approach.

The concepts of this paper have been implemented in *DiaGen*[3] with constraint-based automatic layout based on the constraint solver QOCA by

[3] *DiaGen* can be downloaded from http://www2.informatik.uni-erlangen.de/DiaGen

Chok and Marriott [4]. Their Penguins system also allows to generate free-hand editors, however they do not generate an internal model, but use constraint multiset grammars (CMGs) [3]. However, our hypergraph grammar approach appears to be better suited to the problem since they report a performance that is about two orders of magnitude worse than the performance of *DiaGen* editors on comparable computers. Furthermore, their system cannot support syntax-directed editing since they do not use an intermediate internal model.

On the conceptual level, it is unsatisfactory to some extent to specify syntax-directed editing operations on the less abstract spatial relationship hypergraph instead of the hypergraph model which appears to be better suited for syntax-directed editing (cf. Reker's and Schürr's approach [10]). However, since the mapping from the SRHG to the HGM is non-injective, the approach which has been presented in this paper does not leave much choice if expressiveness should not be sacrificed. However, future work will investigate where specifying syntax-directed editing operations on the more abstract hypergraph model is sufficient.

References

1. Deutsche Norm DIN EN 61131 Teil 3 "Speicherprogrammierbare Steuerungen – Programmiersprachen". Beuth Verlag, Berlin, 1994. In German.
2. R. Bardohl. GenGEd: A generic graphical editor for visual languages based on algebraic graph grammars. In *1998 IEEE Symp. on Visual Languages, Halifax, Canada*. Sept. 1998, pp. 48–55.
3. S. S. Chok and K. Marriott. Automatic construction of user interfaces from constraint multiset grammars. In *1995 IEEE Symp. on Visual Languages, Darmstadt, Germany*. Sept. 1995, pp. 242–249.
4. S. S. Chok, K. Marriott, and T. Paton. Constraint-based diagram beautification. In *1999 IEEE Symp. on Visual Languages, Tokyo, Japan*. Sept. 1999.
5. H. Göttler. Graph grammars and diagram editing. In *Graph Grammars and Their Application to Computer Science*, LNCS 291, pp. 216–231, 1987.
6. H.-J. Kreowski and S. Kuske. Graph Transformation Units and Modules. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, vol. II: Applications, Languages and Tools, pp. 607–638. World Scientific, 1999.
7. M. Minas. Creating semantic representations of diagrams. In *Proc. of the Int. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE'99) at Monastery Rolduc, NL*, 1999. To appear in LNCS, April 2000.
8. M. Minas and G. Viehstaedt. DiaGen: A generator for diagram editors providing direct manipulation and execution of diagrams. In *1995 IEEE Symp. on Visual Languages, Darmstadt, Germany*. Sept. 1995, pp. 203–210.
9. M. Minas and G. Viehstaedt. Specification of diagram editors providing layout adjustment with minimal change. In *1993 IEEE Symp. on Visual Languages, Bergen, Norway*. Aug. 1993, pp. 324–329.
10. J. Rekers and A. Schürr. A graph based framework for the implementation of visual environments. In *1996 IEEE Symp. on Visual Languages, Boulder, CO*. Sept. 1996, pp. 148–155.