Using Graph Grammar to Implement Global Layout for A Visual Programming Language Generation System

Ke -Bing Zhang[1] **Kang Zhang**[2] **Mehmet A. Orgun**[1]

[1]Department of Computing, ICS, Macquarie University, Sydney, NSW 2109, Australia

[2]Department of Computer Science, University of Texas at Dallas
Richardson, TX 75083-0688, USA

{kebing, mehmet}@ics.mq.edu.au

Abstract

VisPro is a general-purpose visual language generation system based on *Reserved Graph Grammar* (RGG). It can express a wide range of diagrammatic *visual programming languages* (VPLs*)*. This paper presents a global layout approach used in the VisPro system. Our approach is grammar-based graph drawing, in which layout rules are embedded in the productions of RGG. Thus, the RGG formalism serves both the visual language grammar and the layout grammar. This approach is appropriate and feasible enough for graph drawing in the area of visual programming languages based on reserved graph grammars.

1 Introduction

For a visual language user, the two most important aspects of a visual program are its physical layout (what the user sees and manipulates), and its meaning (what the user expresses with it). Any implementation of the language has to maintain the correspondence between these two aspects (Andires, Engels and Rekers 1998). A good layout of a visual program could give users a clear view of the syntactic and the semantic relationships amongst the program components.

Layout is a critical component of any visual programming language (VPL) generation system, however it is difficult to implement. First, instead of syntactic and semantic relationships, most graph drawing algorithms are only concerned with the geometrical attributes of nodes and edges in a graph according to aesthetic criteria. Second, the usefulness of a drawing algorithm depends on the type of the graph to be drawn and the application domain. VPL generation systems, such as VisPro, can usually create various types of VPLs, while different layout algorithms are suitable for different types of VPLs. It is therefore not appropriate to use an existing layout algorithm to serve a general-purpose VPL generation system.

Brandenburg (1991) presented a grammar-based graph drawing algorithm, based on a sequential graph rewriting system or graph-grammar to replace some initial graph by productions. The algorithm is efficient in the construction of nice drawings of graphs. Several VPL generation systems, such as VAMPIRE (McIntyre 1994), VLCC (Costagliola, Tortora, Orefice and Lucia 1995), PROGRES (Schürr, Winter and Zündorf 1995) and AGG (Taentzer, Ermel and Rudolf 1999) use graph grammars to specify local layout rules. The main idea of this approach is based on graph transformation, which replaces the left-hand side of a rule by the right-hand side. The left-hand side is a complex graph pattern that describes arbitrary sub-graphs. The right-hand side replaces a sub-graph in the host graph that matches the left-hand side. However, above systems do not use grammar grammars as global layout. This is because the grammar-based graph drawing algorithms do not allow direct specification of global layout criteria such as symmetries (Meyer 1998).

McCreary, Chapman and Shieh (1998) proposed a grammar-based layout approach, that uses a new technique for hierarchical directed graphs in clan-based decomposition. The key idea of this approach is to recognize intrinsic sub-graphs (clans) in a graph by producing a parse tree that carries the nested relationships among the clans. The parse tree of the graph can be given a variety of geometric interpretations. This approach can be used for global layout in a VPL generation system.

VisPro is a general-purpose VPL generation system (Zhang and Zhang 1998[2]) that can generate a wide range of diagrammatic VPLs based on the RGG formalism (Zhang and Zhang 1997). In VisPro we also have embedded layout rules in RGG as layout graph grammar to implement local layout. It is effective on the generation of visual programs (Zhang and Zhang 1999).

To deal with global layout in VisPro, we have developed a layout mechanism based on RGG. In the layout mechanism, the left graph of a production represents a preferred layout and the right graph of the production represents a pattern for layout. In the global layout, the sub-graphs of host graph are computed in isolation and then recombined according to the reserved graph grammar during the parsing process. This paper focuses on the description of global layout in VisPro system. The next section briefly introduces the reserved graph grammar and the presents the basic idea of our approach, followed by its detailed description in the context of RGGs in Section 3. Section 4 demonstrates our layout approach by going through some example visual programs. Section 5 concludes the paper.

2 Background

2.1 Reserved Graph Grammar

Reserved Graph Grammar is a context-sensitive graph grammar, which is complete and explicit in describing the syntax of a wide range of diagrams using labelled graphs (Zhang and Zhang 1997). It is developed based on the *layered graph grammar* (Rekers and Schürr 1997) by using the layered formalism to let the parsing algorithm determine whether a graph is valid in finite steps. It uses an enhanced node structure to simplify the transformation specification and to increase the expressiveness. A simple parsing algorithm with polynomial time complexity can be used for a RGG if the RGG satisfies a particular constraint. A *redex* is a sub-graph of the host graph, which is isomorphic to the right graph of a production in the R-application or to the left graph in L-application. The L-application defines the language of a grammar. The language is defined by all possible graphs which have only terminal labels and can be derived by using L-application from a null graph (i.e. λ). The R-application is used to parse a graph. If the graph is eventually a null after a series of R-applications, the graph is proven to belong to the language.

2.2 Basic Idea

VisPro uses Model, View and Controller (MVC) as its design model (Zhang and Zhang 1998[1]), that uses a protocol for the interaction between the functional modules. The protocol is designed as a combination of an abstract diagram and a concept space. The abstract diagram (Model) describes the structure of a VPL and the concept space specifies the domain concepts for the interaction among the modules. The abstract diagram is created in the parsing process; it is an abstract duplication of the physical diagram. The abstract diagram (Model) is used for parsing the physical diagram (View).

Parsing visual program diagrams in VisPro is based on reserved graph grammars. The parsing process takes two phases: syntactic parsing and semantic parsing.

- Syntactic parsing applies a series of R-applications to a host graph to check whether the program is valid. If an abstract diagram of a visual program is eventually transformed into a null graph (λ), it means that the visual program is valid. This parsing process is behind the physical diagram, and not visible to the user. The physical diagram retains the associated attributes of nodes and edges.
- Semantic parsing produces a result for a diagram by applying a series of L-applications. The result is meaningful only when the diagram is valid. In semantic parsing, one or more actions are executed in each parsing step according to the semantic codes in rewriting rules, and certain attributes of the associated objects (nodes and edges) in the physical diagram of the visual program are changed, such as the appearances of some nodes.

In VisPro, the parsing mechanism can conduct layout operations by performing computations on visual concepts. This is because the appearance of a visual object may be changed dynamically when its visual attributes are modified through the corresponding concept space. Therefore the parse tree of the graph grammar also directs the layout process. In our approach, layout rules form a set of syntax-directed geometrical constraints for visual objects in the physical diagram. The layout rules are embedded in the graph grammar, and are associated with both Model and View. Thus, the graph grammar formalism serves both the visual language grammar and the layout grammar.

Specifying layout rules in VisPro is highly intuitive and easy. When a user specifies a graph grammar for a visual programming language, he/she may specify relative geometrical position of each visual object (node) in the right graph of a production. The specification is similar to specifying graph grammar for a visual language. A parser can be produced according to the graph grammar and the layout rules.

We embed layout rules in the graph grammar so that layout is carried out during the parsing process. During parsing, when a sub-graph, (*redex*) in the host diagram is found, its counterpart in the physical diagram (i.e., View) will be laid out according to the layout rule in the matching right graph. Figure 1 shows the layout workflow in VisPro.

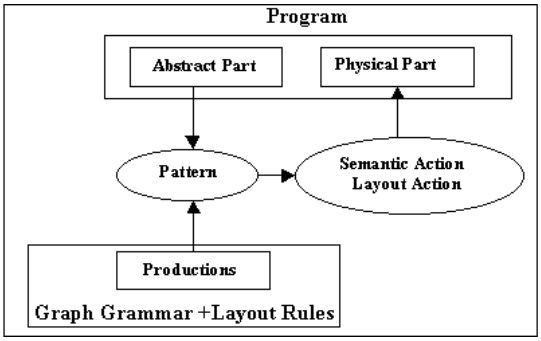

Figure 1: Layout Workflow

2.3 Aesthetic Criteria in Layout

The common aesthetic criteria (Battista, Eades, Tamassia and Tollis 1999) include minimising the number of edge crossings, symmetric drawing wherever possible, minimising the area that the graph uses, no overlap of graph objects, and maximising the symmetry of the overall graph. It is hard to directly specify such criteria in a graph grammar.

We have adapted the techniques of Lai and Eades (1996) and those of McCreary, Chapman and Shieh (1998) to the RGG in the development of the VisPro layout mechanism.

3 The Layout Approach

Our layout approach is based on Brandenburg's (1991) algorithm. It embeds layout rules in the productions of reserved graph grammars. The layout rules in our

approach are specified graphically, rather than textually, through relative geometrical positions.

3.1 Layout Specification

The layout process of a visual program is a series of rewriting transformations that change the geometrical positions of the nodes in the program during parsing. The relative geometrical positions of nodes in right graphs are specified in the attributes of the nodes. The layout specification includes two aspects: (1) the node position and (2) the node direction, in the following forms:

[parsing spec]::=[rewriting rules spec][context spec][layout rule spec]
[layout rule spec]::=[node position spec] [node direction spec]

The layout rules are a set of syntax-directed geometrical constraints for visual objects in the physical diagram. The left-hand sides of rules represent preferred layout and the right-hand sides of rules are patterns for layout.

The layout process is given in the form of an equation $(n_1.a_1, n_2.a_2, \ldots\ldots, n_n.a_n)=l_t(n_1.a_1', n_2.a_2', \ldots\ldots, n_n.a_n)$ where $a_1, a_2, \ldots\ldots, a_n$ are the geometrical attributes of the nodes in a redex before layout, and $a_1', a_2', \ldots\ldots, a_n'$ are those in the redex after layout, and l_t is an arbitrary layout function.

However, not all right graphs of RGG productions carry layout specifications. If the right graph of a production contains more than one node that can be eventually generated from a non-terminal node by applying a series of L-applications, then the right graph may have the layout specification.

For example, as shown in Figure 7, we may notice that in any of productions <1>, <3>, <5> and <7> , the right graph can be generated from the left graph by applying an L-application and in the left graph of each of the productions, there is only one non-terminal node. For production <4>, its right graph can be created by applying L-applications <5> and <4>, so it should carry layout specifications. The right graph of production <6> can be created from the left graph by applying L-application <6> only, however there are two nodes in the left graph. So the right graph of production <6> has no layout specification.

3.2 Layout Aspects

During parsing, when a sub-graph, (*redex*) in the abstract diagram (i.e. Model) is found, its counterpart in the physical diagram (View), called a *super-redex*, will be laid out according to the layout rule in the matching right graph.

Geometrically, after layout transformations a super-redex in a physical host diagram becomes one entity, called a *layout-unit* (like a node in a redex). After layout transformations, the relative geometrical positions between the nodes in a layout-unit are fixed. If one node in the layout-unit is involved in a later layout process and moved, all nodes in the layout-unit will be moved together with that node, i.e., the layout-unit is moved as a single entity.

One node in a right graph is chosen as *focus*, which has a focus label. The other nodes in the right graph, called *non-focuses*, should be specified with direction labels. The node position specification and direction specification in the right graph is used to layout the layout-units in a super-redex in the physical host diagram.

A layout-unit in a super-redex is called an *anchor* if its counterpart node in right graph is focus. The other layout-units in the super-redex are called the *followers*.

The layout rules in the reserved graph grammar include node position specification and node direction specification. The node position specification is a set of geometrical attributes of nodes in a graph (i.e., the positions of nodes in the graph). And the node direction specification is a set of direction labels for the nodes in a constraint graph as illustrated in Figure 2.

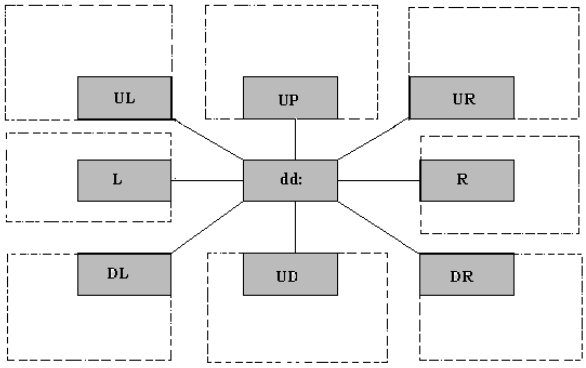

Figure 2: Direction specification in the layout specification

The focus node is labelled “dd”. If a non-focus node is upright above the focus, it is labelled “UP”. Similarly, if a non-focus is at the bottom-left from the focus, a label “DL” should be assigned to the non-focus. In the same way, “L” for left, “R” for right, “UD” for upright under, “DR” for bottom-right, “UR” for top-right and “UL” for top-left, as shown in Figure 2.

Figure 2 illustrates the possible directions of follower nodes (grey boxes) relative to the focus node (grey box labelled dd). Such relative directions may be provided in the right graph of a production. The dashed-boxes in the figure represent layout-units. During the layout process, the followers in a super-redex are positioned according to their layout specifications in the right graph.

For example, if the direction specification of a non-focus is “UP”, the centre point of the follower that maps to the non-focus is moved to the top of the anchor. The distance from the bottom border of the follower to the top border of the anchor is the same as the non-focus to the focus in the right graph. Similarly, if a direction specification is “DL”, the top-right point of the follower should be

moved relatively to the bottom-left point of anchor to the position where the distance between them as the top-right point of the non-focus to the bottom-left point of focus, as shown in Figure 2.

If all nodes in a super-redex are not involved in any layout process (in this case, the super-redex is isomorphic to the right graph), then after layout, the shape of the super-redex is exactly the same as its corresponding right graph, as shown in Figure 3.

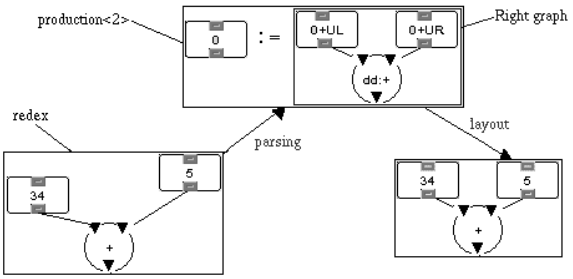

Figure 3: Layout of a *redex*

Applying an R-application with layout rule of a production during the parsing process implies the application of an R-application to the abstract host diagram and layout of the physical host diagram by applying the layout rule in the right graph of the production simultaneously.

After an R-application with layout rule, a laid out super-redex will be merged (recombined) into another super-redex as a layout unit in a later global layout process. If the abstract diagram is transformed into a null graph eventually, then the whole physical host diagram will be laid out as the last super-redex.

4 Examples

We have constructed several VPLs with VisPro. We choose two VPLs *Adder* and *FlowChart* to demonstrate the way in which our layout approach works.

4.1 Adder

Adder has three kinds of nodes: *number*, *adder* and *scaler*. With *Adder*, a user can sum up numbers and visualize the results. It consists a set of productions where box labeled <i> is production i. Figure 4 lists the reserved graph grammar of *Adder*. Layout rules are embedded in the right productions of the graph grammar.

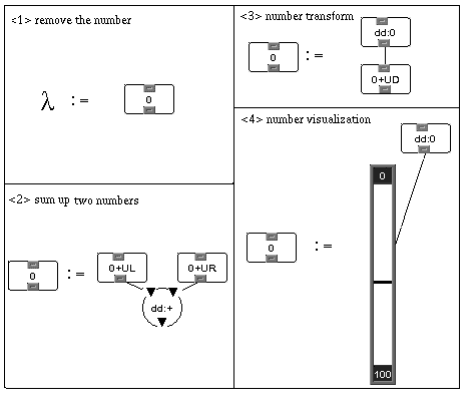

Figure 4: Reserve Graph Grammar with layout rules for *Adder*

Figure 5 shows a visual program written in the *Adder* VPE that was constructed by a user. After compilation, the visual program is properly laid out as shown in Figure 6.

4.2 FlowChart

VPL *FlowChart* offers a visual Flow Chart tool. With *FlowChart* a user can type components such as a number or a string in nodes in a visual program. After parsing the *FlowChart* gives the result according to the user's inputs. The reserved graph grammar with layout rules is shown in Figure 7 where <i> represents production i.

To focus on layout performance in VisPro, we ignored the program semantics in this example. Figure 8 is a diagram of a visual program designed by a user with *FlowChart*. After compilation, the program is shown in Figure 9.

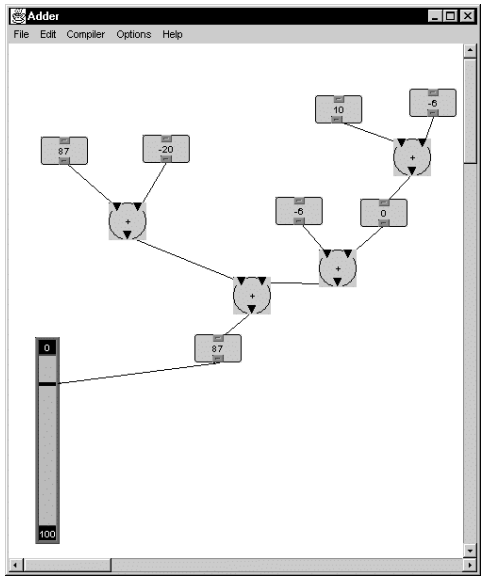

Figure 5: A user designed visual program in *Adder*

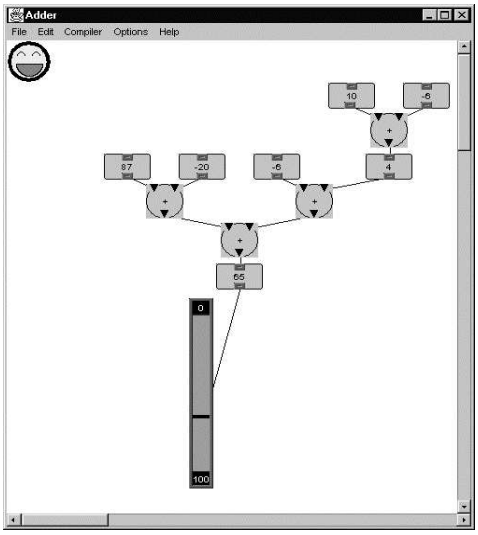

Figure 6: The compiled diagram of the visual program

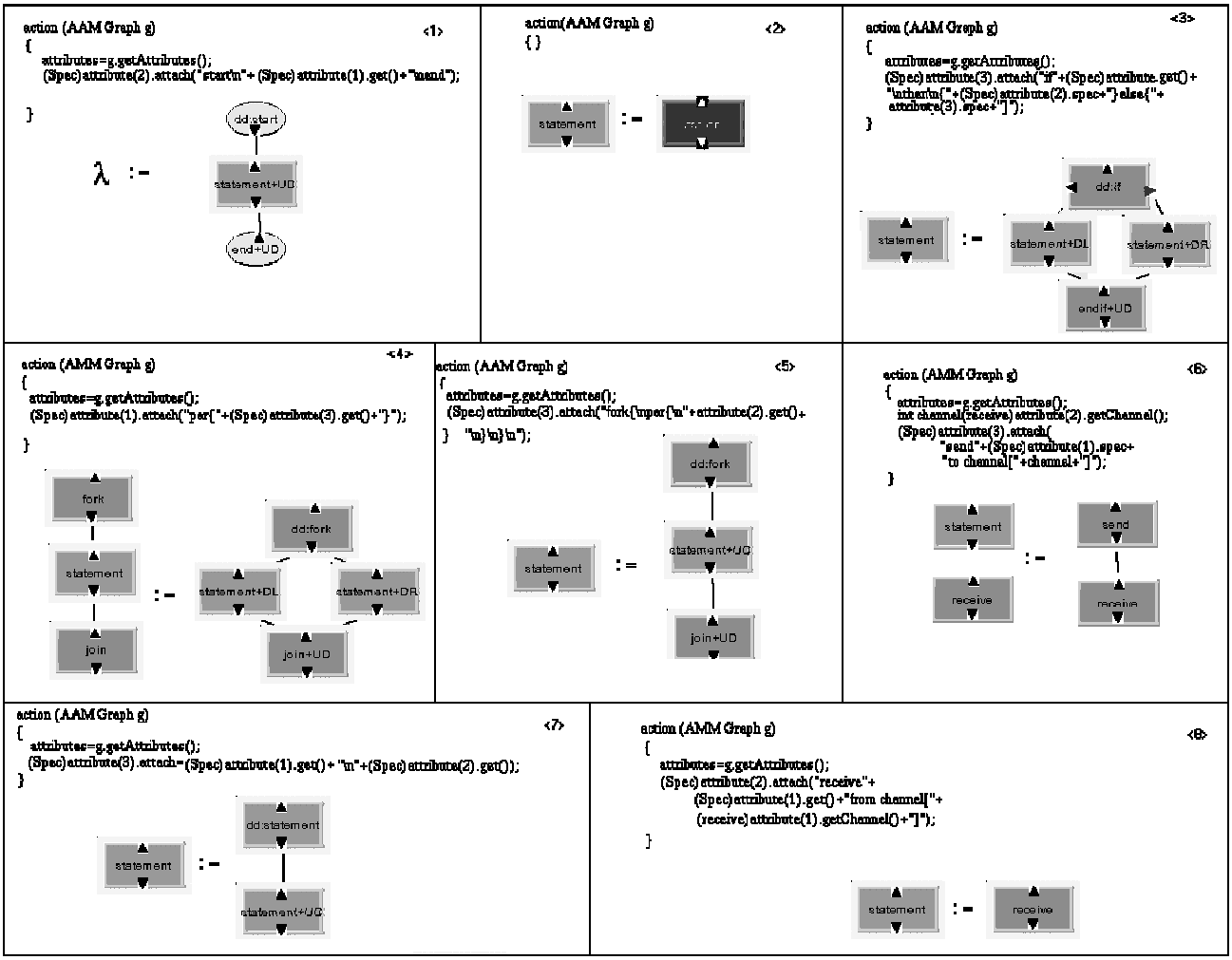

Figure 7: Reserved graph grammar with layout rules and semantics for *FlowChart*

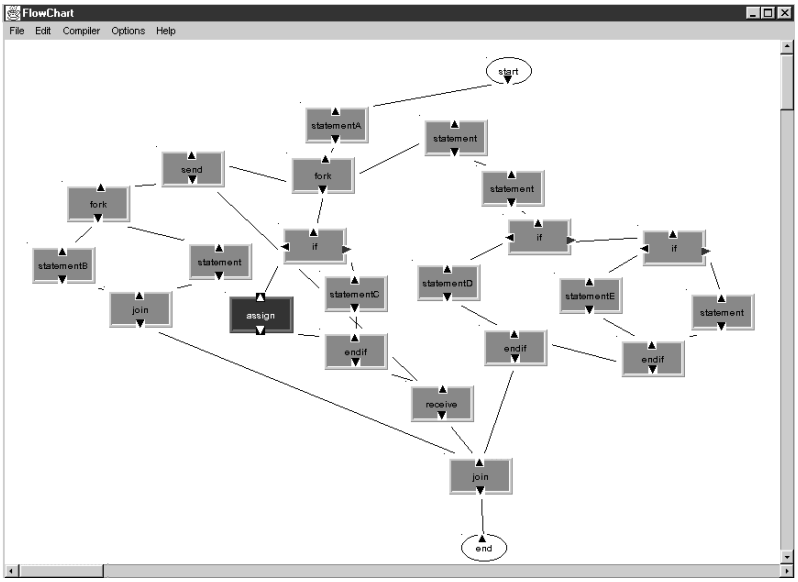

Figure 8: A user designed visual program with *FlowChart* (before compilation)

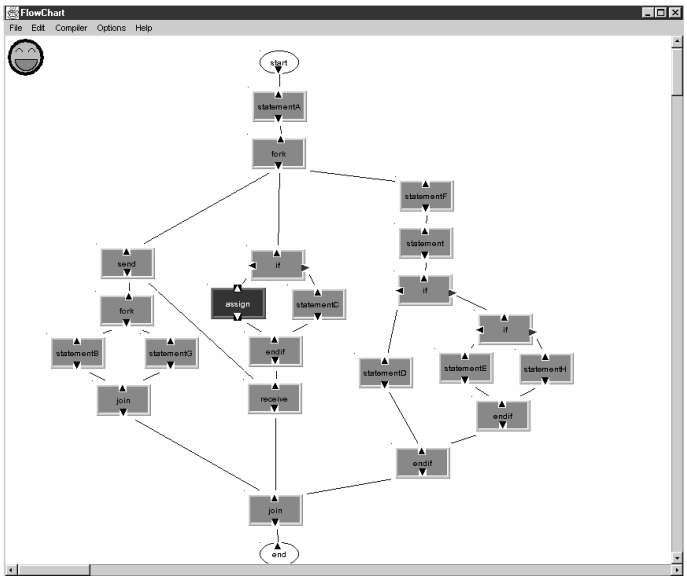

Figure 9: The compiled visual program of the diagram (after compilation)

5 Conclusion

This paper has presented a global layout approach in the VisPro system. It is a grammar-based graph drawing approach by making the reserved graph grammar productions carry layout rules. Thus, the reserved graph grammar is both the graph grammar and the layout grammar for the VPL. When parsing a visual program, layout is performed on the program according to grammar specifications. The layout process is completed when parsing terminates successfully. The approach is therefore highly efficient.

The feature of layout rules embedded in graph grammar makes layout process general enough for a visual programming language generation. This approach is suitable and feasible enough for layout graphs of visual programs in VisPro. It is flexible and highly intuitive. With these advantages, a VPL designer can develop a graph grammar for a visual programming language easily and clearly in the VisPro system.

6 Reference

ANDRIES, M., ENGELS, G. and REKERS, J. (1998): How to represent a visual specification. In *Visual Language Theory*. 245-260. MARRIOTT, M. and MEYER, B. (eds). Springer Verlag.

BATTISTA, G. Di, EADES, P., TAMASSIA, R. and TOLLIS, I.G.(1999): Graph drawing algorithms for the visualization of graphs. Englewood Cliffs, Prentice-Hall.

BRANDENBURG, F. J. (1991): Layout Graph Grammars: the Placement Approach. *Lecture Notes in Computer Science, Graph Grammars and Their Application to Computer Science* **532**:144-156. Springer Verlag, Berlin.

COSTAGLIOLA, G., TORTORA, G., OREFICE, S. and LUCIA, A.D. (1995): Automatic Generation of Visual Programming Environments. *IEEE Computer*, **28**(3):56-67.

LAI, W. and EADES, P. (1996): Structural Modeling of Flowcharts. In *Software Visualisation*, Series on Software Engineering and Knowledge Engineering. **7**:232-243. EADES, P. and ZHANG, K. (eds). World Scientific Co.

McCREARY, C. L., CHAPMAN, R., and SHIEH, F-S. (1998): Using Graph Parsing for Automatic Graph Drawing. *IEEE Transactions on Systems, Man and Cybernetics* **28**(5): 545-561

MCINTYRE D. W. (1994): Design and Implementation with Vampire. In *Visual Object-Oriented Programming*. 129-159. BURNETT, M., GOLDBREG, A. and LEWIS, T. (eds). Manning Publish Co.

MEYER, B. (1998): Competitive Learning of Network Diagram Layout, *Proc.VL'98-1998 IEEE Symposium on Visual Languages*, Halifax, Canada, 56-63, IEEE CS Press, Los Alamitos, USA.

REKERS, J., and SCHÜRR, A.(1997): Definiting and Parsing Visual Languages with Layered Graph Grammars. *Journal of Visual Languages and Computing* **8**(1):27-55.

SCHÜRR, A., WINTER, A., and ZÜNDORF, A. (1995): Visual Programming with Graph Rewriting Systems. *Proc. VL'95-11th Int. IEEE Symop. on Visual Languages*, Darmstadt, Germany, 326-335, IEEE CS Press, Los Alamitos, USA.

TAENTZER, G., ERMEL, C. and RUDOLF, M. (1999): The AGG-approach: Language and tool environment. In *Handbook of Graph Grammars and Computing by Graph Transformation, Applications, Languages and Tools*. 2: 551-604. EHRIG, H., ENGELS, G., KREOWSKI, H-J. and ROZENBERG, G. (eds). World Scientific Co.

ZHANG D-Q. and ZHANG K. (1997): Reserved Graph Grammar: A Specification Tool for Diagrammatic VPLs. *Proc. VL'97-13th IEEE Symposium on Visual Languages*, Capri, Italy, 284-291, IEEE CS Press, Los Alamitos, USA.

ZHANG D-Q. and ZHANG K. (1998[1]): On the Design of A Generic Visual Programming Environment. *Proc.VL'98-1998 IEEE Symposium on Visual Languages*, Halifax, Canada, 88-89, IEEE CS Press, Los Alamitos, USA.

ZHANG D-Q. and ZHANG K. (1998[2]): VisPro: A Visual Language Generation Toolset, *Proc.VL'98-1998 IEEE Symposium on Visual Languages*, Halifax, Canada, 195-202, IEEE CS Press, Los Alamitos, USA.

ZHANG K-B. and ZHANG K. (1999): An Incremental Approach to Graph layout Based on Grid Drawing, *Proceedings of the Third Workshop on Software Visualisation* (SoftVis'99). 41-50, University of Technology, Sydney.