Compiled Visual Programs by VisPro

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

[1]Department of Computing, ICS, Macquarie University, Sydney, NSW 2109, Australia
{kebing, mehmet}@ics.mq.edu.au

[2]Department of Computer Science, University of Texas at Dallas
Richardson, TX 75083-0688, USA
kzhang@utdallas.edu

Abstract

VisPro is a general-purpose visual language generation system based on Reserved Graph Grammar (RGG). It is also the execution environment of visual programming languages (VPLs) developed under the VisPro system. However, due to the lack of compilation facilities, VisPro could only generate simple interpreted VPLs. We have developed a compilation mechanism for VisPro to implement compiled visual programs (VPs). The compiled visual programs are the executable diagrams that can be executed repeatedly. We present our approach in details in this paper.

1 Introduction

According to the program execution, textual programming languages can be roughly classified into two types: *interpreted languages* and *compiled languages.*

The execution of an interpreted language program depends on an interpreter program that reads the source code and translates it on the fly into computations and system calls dynamically with the syntactical parsing and semantic parsing of the program.

However, interpreted languages, for example Basic, do not produce target code (machine code) files, therefore, the source programs have to be re-interpreted and re-executed each time. Interpreted languages tend to be slower than compiled languages. On the other hand, they tend to be easier to program.

The execution of a compiled language program, for example C, is more complicated than an interpreted one. First, the source program is translated into an executable target machine code by a compiler. Then users can run the executable code directly without looking at the source code again. Compiled languages tend to run quicker than interpreted languages, but the executable code generation is complicated. The advantage of compiled languages is not only their high speed but also their once compiled and repeatable execution feature.

The concept of interpreted and compiled in visual programming languages (VPLs) is similar to the textual language. However most visual languages produced by visual language generation systems, such as GenGEd (Bardohl, R.1998) Penguins (Chok and Marriott 1998) VLCC (Costagliola et al.1995), DiaGen (Minas and Viehstaedt 1995) and VL-Eli (Kastens and Schmidt 2002), are interpreted. This is because those visual languages are simple and most of them are used as interactive systems.

VisPro is a general-purpose VPL generation system (Zhang and Zhang 1998, Zhang et al. 2001) that can generate integrated diagrammatic VPLs based on the reserved graph grammar (RGG) formalism (Zhang and Zhang 1997).

2 Reserved Graph Grammar

A graph grammar has a set of graph rewriting rules. Each rule has two graphs, which are called *left graph* and *right graph*. It can be applied to another graph (called *host graph*) in the form of an *L–application* or *R–application*.

A *redex* is a subgraph in the host graph which is isomorphic to the right graph in an R–application or the left graph in an L–application. A rule's L–application to a host graph is to find a *redex* in the host graph of the left graph of the rule and replace the redex with the right graph of the rule. An R–application is a reverse replacement of its corresponding L-application (i.e. from the right graph to the left graph).

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 transformed into a null graph after a series of R-applications, the graph is proven to belong to the language.

A Reserved Graph Grammar is a context-sensitive graph grammar, which is complete and explicit in describing the syntax of diagrams using labelled graphs (Zhang and Zhang 1997). 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.

The VisPro graph rewriting system is a tool for graph transformation based on the attribute graph grammar. Its graph rewriting rules specify the high-level graph transformation, and the low level code associated with the rules specify the attribute computations. An attribute in the VisPro graph rewriting system is an object in the

object-orient formalism. An object can be active during the parsing and can change its attributes. This is useful in interpreting a high level diagram.

The VisPro graph rewriting system accepts a set of graph rewriting rules as its reasoning knowledge. The system can then parse a diagram and produce results by interpreting the diagram with the rules.

3 Visual programs execution in VisProa

3.1 Interpretation of VPLs in VisPro

The concept of interpretation in VisPro is similar to that in a textual language. For an interpreted VPL, the syntactical and semantic specifications of the VPL are usually specified in the same set of graph rewriting rules in RGG. The interpreter of the VPL performs semantic transformations on the matching redexes according to the graph rewriting rules during syntactical parsing.

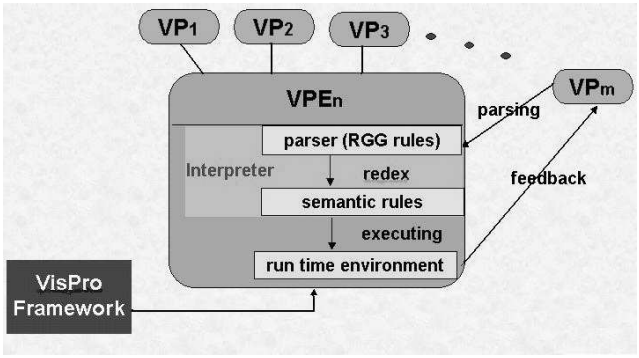


Figure 1. Visual programs execution of interpreted VPLs.

When a redex is found in an abstract graph of the visual program diagram, i.e., host graph during each syntactical parsing step, the semantic execution is performed immediately on the redex according to the semantic (action) codes in the graph rewriting rules, and certain attributes of the associated nodes and edges in the visual program diagram are changed, such as the appearances of some nodes. When the syntactical parsing of the program terminates, the semantic execution also completes. No target code is produced in this process. The parsing and execution process of the interpreted visual program (VP) in VisPro is illustrated in Figure1.

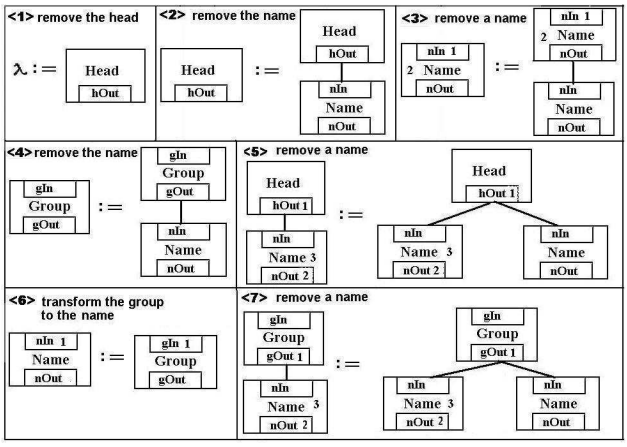


Figure 2. The reserved graph grammar for LookUp

In VisPro, the semantic execution sequence of an interpreted visual program is specified in the RGG of the VPL by a marking mechanism. For example, Figure 2 shows the reserved graph grammar of VPL LookUp, in rule <3>, where node *name* with "2" and sub-node *nIn* with "1" are marked, others are unmarked.

The semantics of LookUp is to find the organizational structure people. If a person has children nodes, the person is the supervisor of the people in the children node and the name of the person will be marked a series of "*". The number of star characters depends on the amount of people under the person. For example, as shown in Figure3, there are tree people under "R.ENGEL". Thus, after the program execution, his name should have three stars "*", as shown in Figure 4.

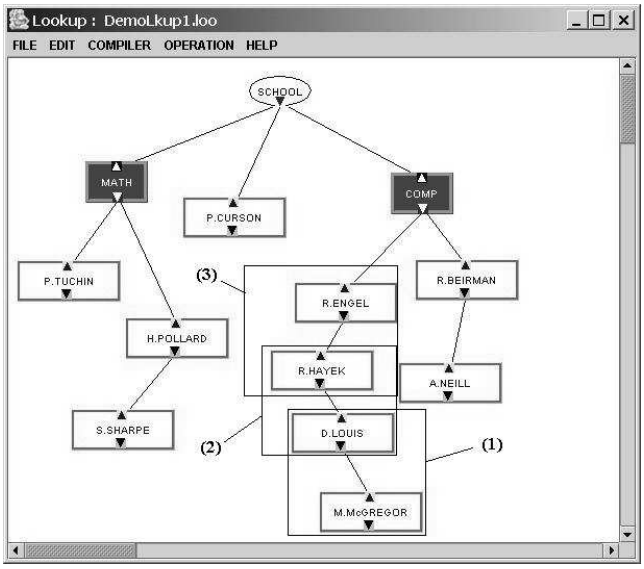


Figure 3. A visual program of *LookUp* (Before parsing)

According to the semantics of LookUp, the execution sequence of the program as shown in Figure 3 should be sub-graphs (1) then sub-graph (2), and finally sub-graph (3). Otherwise, it will produce an incorrect result. The execution sequence is determined by the marked (sub-)nodes of LookUp, as shown in rule <3> in Figure 2, because an unmarked object can be matched only when all of its edges are inside the redex (Zhang and Zhang 1997).

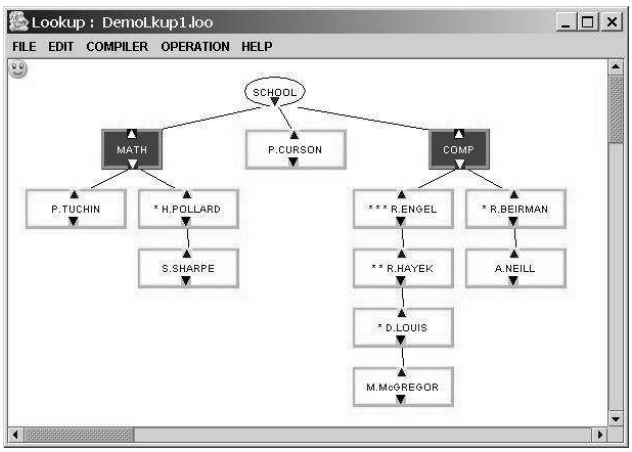


Figure 4. The executed visual program of Figure3.

Figure 4 is the result diagram of Figure 3 after parsing. The semantic execution is applied to each redex with the syntactical parsing of the redex. This is typical execution of an interpreted language.

However, in the process of parsing an interpreted visual program in VisPro, no parse tree is produced. Without a parse tree, using the marking mechanism to specify the semantics execution order sometimes does not work with some reserved graph grammars.

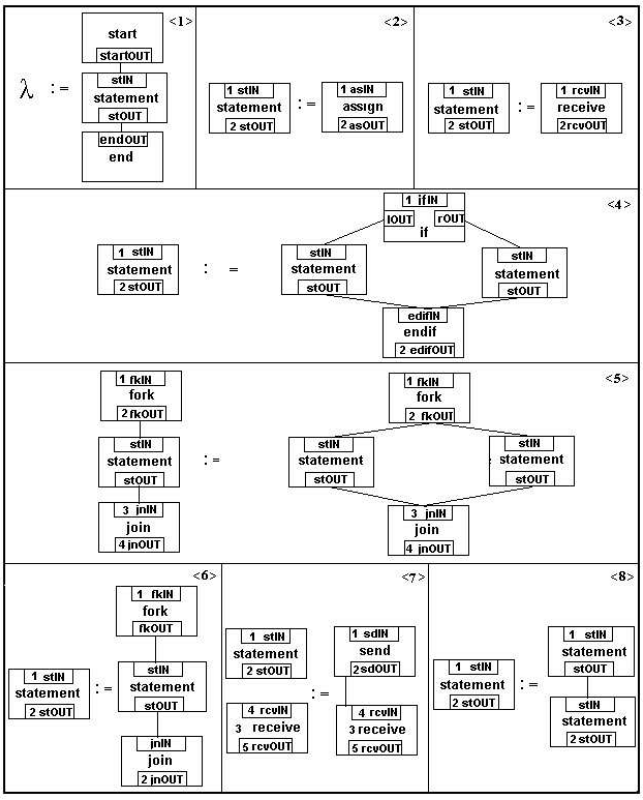


Figure 5. The reserved graph grammar of Flowchart

For example, Figure 5 illustrates the reserved graph grammar of VPL Flowchart, and Figure 6 gives a visual program of the VPL Flowchart. According to the semantics of Flowchart, the dataflow should be top-down. This means that the semantic execution should apply sub-graph (1) then sub-graph (2). See Figure 5, rule <8> matches with sub-graph (1) and sub-graph (2) in Figure 6. It is clear that either sub-graph (1) or sub-graph (2) can be redexes of rule <8> because a marked node can be matched when it has any number of edges.

However, the mark nodes in the graph rewriting rules of Flowchart cannot guarantee graph (1) first then graph (2) application order, since syntactical parsing order depends on the physical generation order of the nodes in the host graph for the same rewriting rule. Therefore, using the marking mechanism to specify the semantic execution order of the visual programs may produce incorrect execution results.

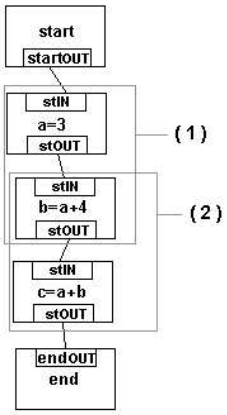


Figure 6. A visual program of Flowchart

To overcome this limitation of RGGs, we have introduced an ordering mechanism to RGGs to specify the execution sequences of visual programs (Zhang et al. 2002). Also, we have extended the parser to produce parse trees for sophisticated VPLs (Zhang et al. 2003).

3.2 Compiled visual program generation

The parsing process in VisPro takes two phases: syntactical parsing and semantic parsing.

Syntactical parsing applies a series of R-applications to the host graph of a visual program to check whether the program is valid. If the host graph is eventually transformed into a null graph (λ), it means that the visual program is valid. If the syntactical parsing is successful, the compilation mechanism we developed for VisPro can produce a parse tree during this phase.

Semantic parsing checks the source program for semantic errors and gathers type information for the subsequent code generation phase by applying a series of L-applications according to the parse tree produced in syntactical parsing. The result of semantic parsing is meaningful when the semantic parsing is successful.

Finally, the compiler of VPL generates an executable code for the visual program. The executable code of the visual program contains the hierarchical structure of the parse tree produced in the syntactical parsing phase and gives the execution sequence of the objects in the visual program.

3.3 Compiled Visual Program

The result of a compiled visual program in VisPro is an executable diagram which contains two parts: a user source visual program diagram and its executable code. The executable code is in high-level text, instead of machine code. Such a simple executable code of a compiled visual program is useful, especially for complicated visual programs. This is because the two-dimensioned nature of visual languages and syntactical parsing of visual program diagrams are time consuming.

The run-time environment of a VPL in VisPro can use the executable code to indicate the execution sequence of visual program source diagrams and apply the actions to

the visual program diagrams. The execution process is illustrated in Figure7.

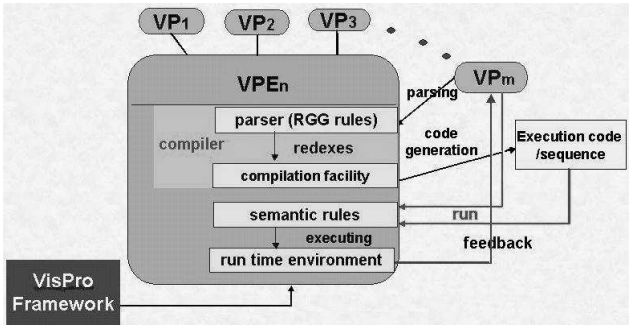


Figure 7. The execution of compiled visual programs.

Recently more researchers have been interested in visual programming in distributed environments, especially Web-based applications (Hoppe et al. 2000, Mosconi et al 2001, Mosconi et al. 2003). VisPro is not only a visual programming language generation system, but also a visual program execution environment. Therefore, with the simple executable code, VisPro has potential to be developed as a distributed VPL generator.

4 An Example

To demonstrate our system, we specified very simple semantics for VPL FlowChart. In this VPL, the value of a node is transformed from one to another if there is an edge between them. The value can be shown on the "statement" nodes only.

Figure 8 illustrates a visual program designed in VPL FlowChart, where diagrams (1), (2) and (3) show the original visual program, the visual program after syntactical parsing and the visual program after execution respectively.

5 Conclusion

This paper has proposed an approach to generating compiled visual programs in the VisPro system. The targets of compiled visual programs are the executable diagrams and can be executed repeatedly. The executable code of compiled visual programs is intuitive. The compiled visual programs in VisPro have more advantages over interpreted VPLs, especially for complicated visual programming languages.

6 Reference

Bardohl, R. (1998): GenGEd - A Generic Graphical Editor for Visual Languages based on Algebraic Graph Grammars. in *Proc. IEEE Symposium on Visual Languages (VL'98)*, Sept.1998, Halifax, Canada, pp. 48-55.

Chok, S. and Marriott, K. (1998): Automatic Construction of Intelligent Diagram Editors. *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1998, pp.185-194,

Costagliola, G. Tortora, G. Orefice, S. and Lucia, A.D (1995): Automatic Generation of Visual Programming Environments, *IEEE Computer*, 28(3), March, 1995, pp.56- 66

Hoppe, U. Ganer, K. Mhlenbrock, M. and Tewissen, F. (2000): Distributed Visual Language Environments for Cooperation and Learning: Applications & Intelligent Support. *Journal Group Decision and Negotiation*, 9(3), 2000.

Kastens, U. and Schmidt C. (2002): VL-Eli: A Generator for Visual Languages. *Second Workshop on Language Descriptions, Tools and Applications (LDTA 2002)* Grenoble, France, 13 April 2002.

Minas, M. and Viehstaedt, G. (1995): DiaGen: a generator for diagram editors providing direct manipulation and execution of diagrams. *Proc.11th IEEE Symp. on Visual Languages*, September 5--9, 1995. Darmstadt, Germany,. IEEE Computer Society Press, Los Alamitos, CA. pp. 203-210

Mosconi. M. and Porta, M. (2001): Programming Web Applications through a Data-Flow Visual Approach. *Journal of Applied Systems Studies (JASS)*, Vol. 2, No. 3, 2001.

Mosconi M, Ottelli M. D. and Porta, M. (2003): Alligator, a Web-based Distributed Visual Programming Environment. *The 12th International World Wide Web Conference* Budapest, Hungary, 20-24 May 2003.

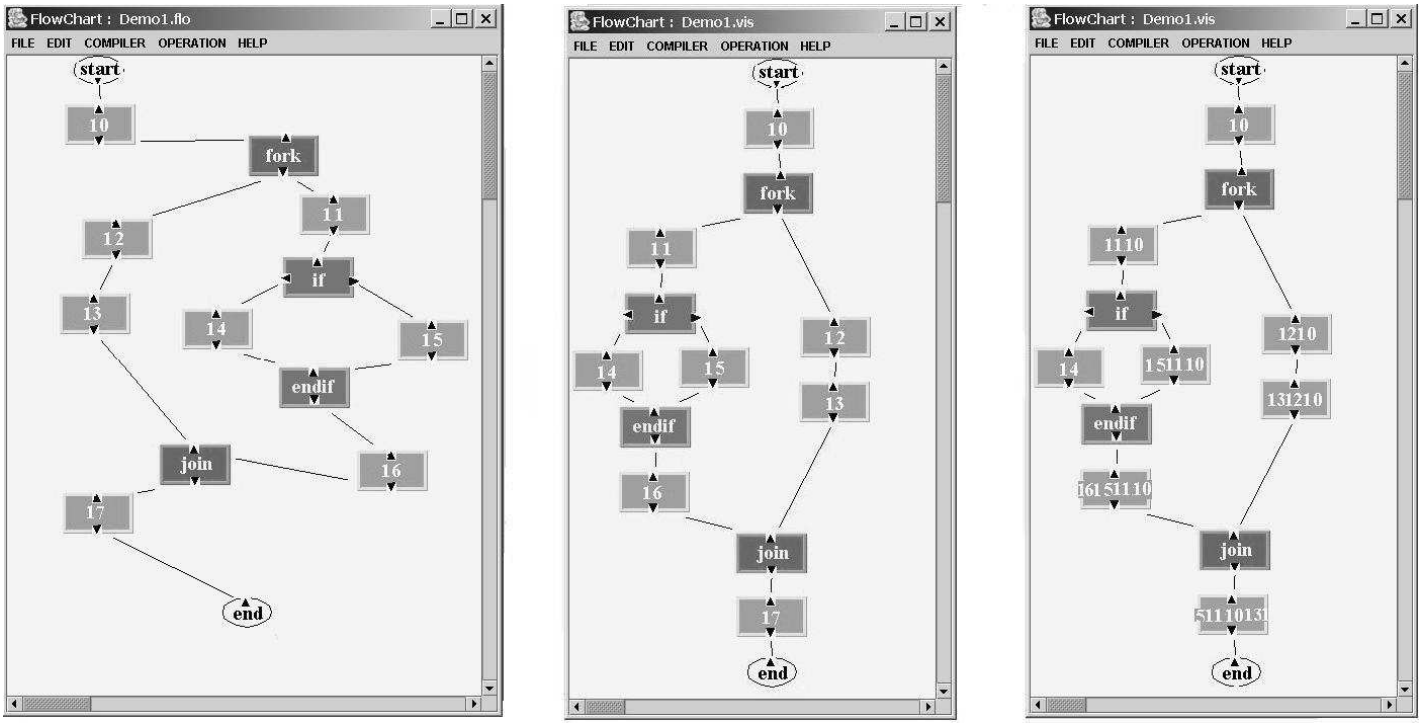
Zhang, K.B. Orgun. M.A. and Zhang, K. (2002): Visual Language Semantics Specification in the VisPro System", *Pan-Sydney Area Workshop on Visual Information Processing*, Sydney Australia, November 2002.

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, 23-26 September, 1997, IEEE Computer Society Press, Los Alamitos, USA, pp.284-291.

Zhang, D-Q. and Zhang, K.(1998): VisPro: A Visual Language Generation Toolset, *Proc.VL'98-1998 IEEE Symposium on Visual Languages*, Halifax, Canada, 1-4 September 1998, IEEE CS Press, Los Alamitos, USA, pp.195-202

Zhang, K., Zhang, D-Q. and Cao, J. (2001): Design, Construction, andApplication of a Generic Visual Language Generation Environment. *IEEE Transactions on Software Engineering*, April 2001,vol.27, no.4, pp. 289-307.

Zhang, K.B., Zhang K. and Orgun, M.A. (2003): Semantic Specifications in Reserved Graph Grammars" *The Ninth International Conference on Distributed Multimedia Systems (DMS'2003)*, Miami, Florida, USA September 2003.



(1) original (2) parsed (3) executed

Figure 8. Compilation demo I of VPL Flowchart