Crossroads
The ACM Student Magazine

Programs worth 1000 words

Visual languages bring programming to the masses

*by**Lorrie Cranor** and **Ajay Apte***

Over the past decade visual programming languages have advanced from laboratory toys to commercial products. While still far from the mainstream, visual languages are emerging as important special purpose languages, particularly for programming tasks generally performed by non-programmers.

But the idea of representing programs with pictures is not new. For many years programmers have used graphical aids to help design and document programs. Flow charts and block diagrams help illustrate algorithms and data structures, while graphical symbolic debuggers and object-oriented graphical programming environments help programmers organize and debug large programs. Visual representations of problems and solutions are useful because they are often more intuitive than textual representations. In addition, visual representations transmit data through ``the largest bandwidth communication channel humans have'' **[1]**.

Most people learn to communicate using pictures at a very young age. Pictures are often a more powerful means of communication than words because they can convey much meaning in a concise unit of expression. In addition, pictures do not have language barriers. When properly designed, they are understood by people regardless of what language they speak **[3]** . A computer program that uses two-dimensional diagrams and icons is often more easily understood by end users than a conventional C or Pascal program. For non-programmers who only wish to know enough about programming to customize their application programs and computer environments, graphical or visual programming languages are easier to learn and use than

conventional textual programming languages.

There have been many recent projects to develop visual languages for user interface specification. This work has resulted in the evolution of three broad categories of visual programming systems: those that use conventional visual programming paradigms, those that use programming by example, and those that use form-based programming. Conventional systems provide a set of visual primitives and control structures which are used for program construction. Systems that use programming-by-example automatically generate the program code by making inferences based on the interaction between the user and the application. Form-based programming systems allow the programmer to first specify the format of a graphical user interface, and then define the functions associated with each component.

Many traditional visual programming languages use a *data flow* paradigm in which data flows through a series of connected boxes. Each box acts as a filter that may transform the data before passing it on to the next box. Boxes may represent variables, arithmetic operations, mathematical functions, or entire programs. The data flow model is intuitive because it models a physical transfer of data from filter to filter. In addition, the data flow model does not require the use of named variables, thus making it suitable for young children and other non-programmers who are not familiar with the somewhat abstract concept of variables. Figure 1 shows a dataflow representation of the addition of two numbers.

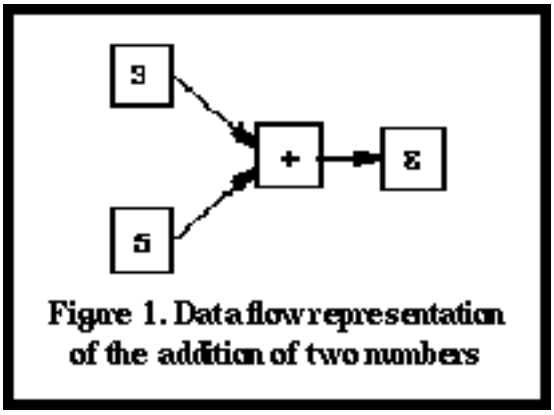

Figure 1. Data flow representation of the addition of two numbers

Prograph is a commercially available visual programming language that supports a dataflow specification of program execution. It also provides an object-oriented application building toolkit. In Prograph, classes are represented as icons that can be viewed using a graphical class browser. The Prograph system includes an editor for creating classes and programs, an interpreter for executing and debugging programs, and an ``Application Builder'' which is used for constructing and testing user

interfaces. An important feature of Prograph is that the editor and interpreter can run concurrently. Therefore, the user can pause a running application, modify the elements of the user interface, and resume the application with new functionality. Users and system events can trigger calls to user methods which may not yet exist but can be created and edited ``on the fly.'' The Application Builder is similar to the interface construction tools provided with other products such as Visual Basic.

Laboratory Virtual Instrument Engineering Workshop (LabVIEW) is another commercial visual programming system. LabVIEW combines structured programming and data flow programming paradigms and embeds them in a graphical editing and execution system. It uses a virtual instrument metaphor in which a program is viewed as a hierarchy of instrument-like modules containing interactive front panels and block diagrams.

Show and Tell

The Show and Tell Language (STL) is a data flow based, general purpose programming language originally designed for school children **[2]**. As with most data flow models, the directed graph of boxes and arrows, called a *boxgraph*, serves as the semantic base for STL. In STILL, boxes represent functions, constraints, variables, iterators, or containers of inconsistence. Arrows represent the flow of data between boxes. In the boxgraph notation, boxes may be hierarchically nested and arrows may point from a box at any level in the hierarchy to another box at any other level, as long as no cycle is created in the boxgraph.

STL uses the concept of consistency for program control, emulating an IF-THEN construct without the introduction of specific flow control ideas. A boxgraph becomes inconsistent when there is a conflict of some type among the data flowing into the box. Figure 2 shows an example arithmetic problem which is designed to give an answer only when two expressions are equal. In the example, the expressions (3+5 and 3X5) are not equal, and thus nothing flows out of the shaded consistency box. If the expressions 2+2 and 2X2 were given instead, the number 4 would appear in the answer box.

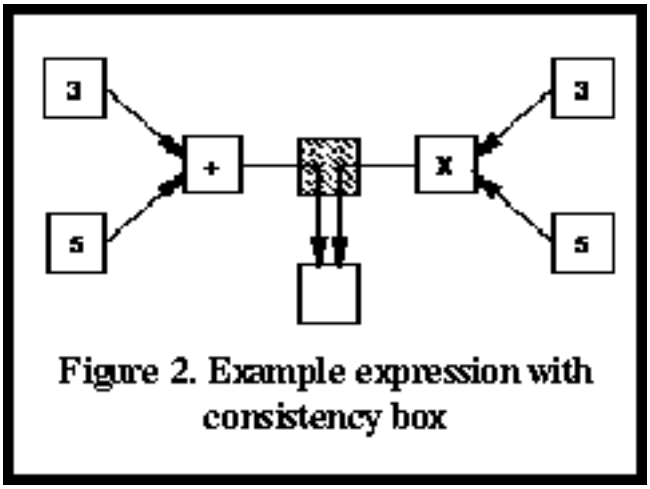

Figure 2. Example expression with consistency box

STL provides a number of box types to perform arithmetic operations, iterations, higher order functions, and database construction. Figure 3 shows an example STL program for recursively computing factorials. Notice that when 0 is given as input, the problem reduces to the base case. All other positive inputs first reduce the problem to the base case, and then build it back up again recursively.

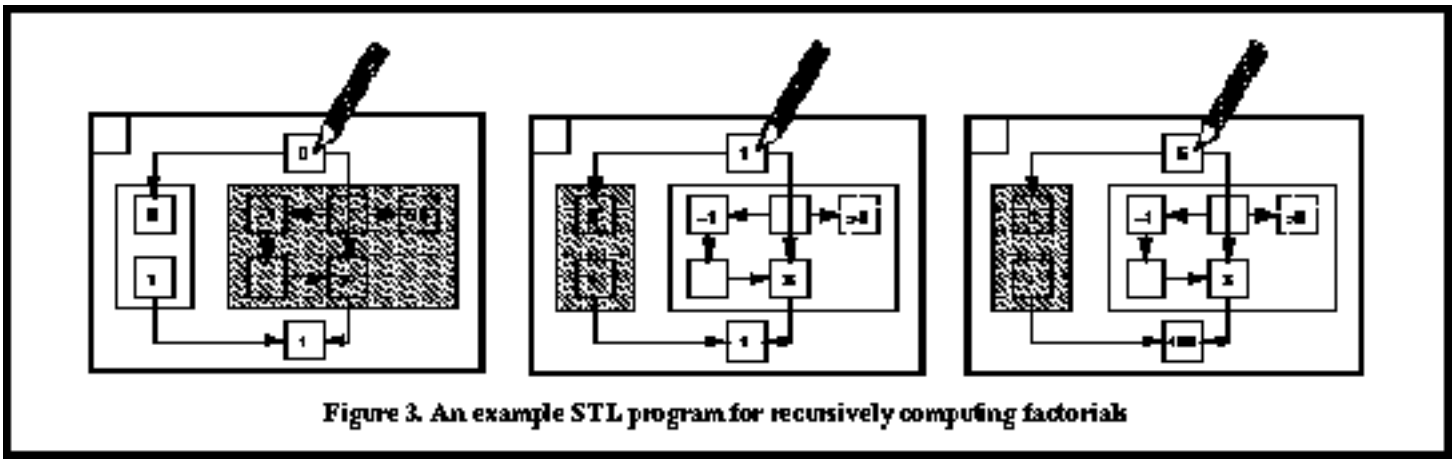
Figure 3. An example STL program for recursively computing factorials

Future Directions

As already noted, visual languages are particularly useful for special purpose applications and for programming tasks generally performed by nonprogrammers. Commercial visual languages such as Prograph are being used by doctors and business people to construct databases. The commercial visual programming environment LabVIEW is being used for laboratory automation by engineers with little or no traditional programming experience.

In the long term, visual languages may also prove to be useful tools for expert programmers who do operating systems work and develop large application programs. Already, expert programmers are turning to graphical user interfaces and symbolic

debuggers to make their work easier. And as visual languages develop, they may enable the development of visual software engineering environments as well.

But visual languages are not without limitations. Some of the characteristics that make visual languages powerful and easy to learn also limit their usefulness as general purpose languages. Visual languages are easy for nonprogrammers to understand because they are very concrete. However, because these languages are so concrete, it is difficult to develop an abstract semantic model for them. This limits the use of visual languages for large programming problems that benefit from the liberal use of abstraction. In addition, the non-linearity of visual languages makes it difficult to build formal grammars and compilers for them. While text strings only allow concatenation in two places - before a character and after a character - visual languages allow an infinite number of concatenation options, as shown in Figure 4. Because the visual language ``alphabet'' is infinitely large, attempts at developing visual grammars using textual grammars as models have been unsuccessful.

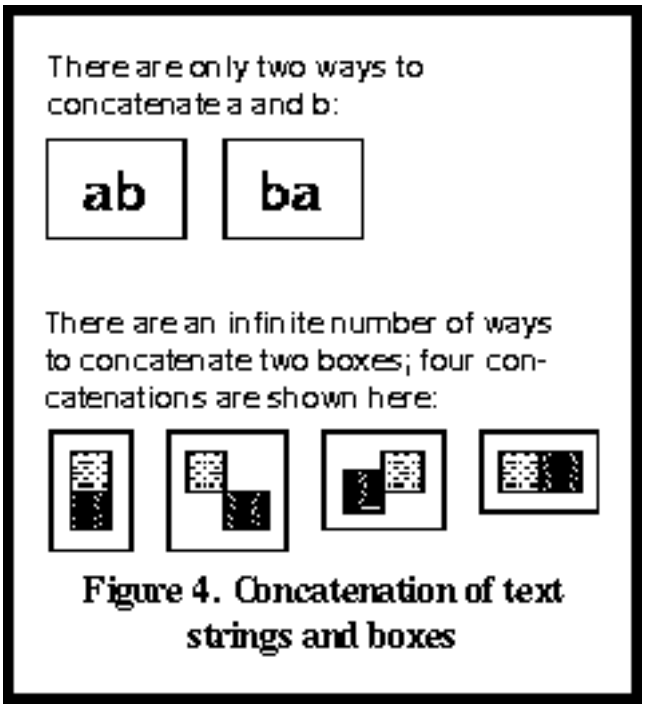

Figure 4. Concatenation of text strings and boxes

Another problem with most existing visual languages is that they are hard to understand when used to create large programs. In data flow model languages, for example, large programs often result in unwieldy tangles of arrows that are difficult to trace and hard to break down into component parts.

Despite these limitations, there is growing interest in visual language development and use. Besides serving as a new paradigm for programmers, visual languages may help

make computers accessible to lay people. In the future, as home appliances become increasingly programmable, wireless pen based computers and visual languages may replace push buttons and remote controls. And, as visual languages improve, they may eventually offer an easy way to program a VCR.

References

1. Clarisse, O. and S-K. Chang. VICON a visual icon manager. In *Visual Lanugages.* eds. S-K. Chang, T. Ichikawa, and P. Ligomenides, 151-190. New York: Plenum Press, 1986.

2. Kimura T.D.,J.C. Choi and J.M. Mack. *A Visual Language for Keyboardless Programming.* Technical Report WUCS-86-6, Department of Computer Science, Washington University, St. Louis, June 1986.

3. Shu, N.C. *Visual Programming.* New York: Van Nostrand Reinhold, 1988.

Ajay Apte graduated from Washington University in 1994 with a Master of Science degree in Computer Science. He currently works as a software consultant for Systems Programming Ltd. His interests include visual programming and user interface design.