Improving Visual Programming Languages for Multimedia Authoring[1]

John F. Koegel and Jesse M. Heines

Department of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854

{koegel,heines}@cs.uml.edu

ABSTRACT

Visual programming languages are emerging as an important paradigm for development of interactive multimedia presentations. We are actively developing and using such languages and have identified several criteria for evaluating the user interface for such tools. These criteria include: 1) support for the authoring process, 2) icon set semantics, and 3) usability of the interface. In this paper we compare two commercial authoring tools and discuss visual programs used for our evaluation.

1. Overview

1.1 Multimedia Authoring

The creation and editing of interactive computer-based presentations which combine text, graphics, audio, and video is called multimedia authoring. Contemporary tools provide both media editing and multimedia composition as well as more specialized services including database and file access, courseware support, and extensibility. Such tools make it possible to design sophisticated presentations, but are more difficult to use than previous menu and text systems for several reasons:

- The more powerful authoring environments require more learning time
- The creation and integration of animations, video, and audio is a more elaborate process and is typically less familiar to authors than text and graphics composition

We have been actively developing and using multimedia authoring tools for a number of years. This paper summarizes our observations and experience with two commercial multimedia authoring packages and provides the following results:

- Detailed criteria for evaluating visual programming interfaces for multimedia authoring, including icon set semantics and support for the authoring process
- Example iconic programs for comparing different multimedia visual programming tools

[1] This paper was presented at the ED MEDIA 93 World Conference on Educational Multimedia and Hypermedia in Orlando, Florida, June, 1993, and was published in Proceedings of ED-MEDIA 93, ed. H. Maurer, pp. 286-293, Association for the Advancement of Computing in Education, Charlottsville, Virginia, USA.

- General recommendations for improving the design of such systems' user interfaces

Multimedia authoring tools will become a significant enabling tool for courseware and other interactive presentations. However, the development of such materials with the best tools available today requires significant expertise, equipment, and time. The goal of this paper is to further advance the usability of such tools.

1.2 Synopsis of Authoring Paradigms

There are at least fifty commercial packages for developing computer-based training or interactive presentations. Many of these provide audio-visual device control for multimedia delivery. Listed below are the three of the most common paradigms for multimedia authoring.

- **Outline**: An outline of the presentation is constructed in a text-based outlining editor. Each outline entry can be expanded into a presentation screen which incorporates graphics, text, and interaction.

- **Visual Programming**: A set of icons are arranged in a graph which specifies the interactions and control path for the presentation. The functionality associated with each icon can be modified using associated menus and editors. Typically a simple text language is available for performing calculations within an icon.

- **Scriptware**: Graphics, text, and other media editors are combined with an integrated and special purpose programming language. Programs (scripts) define the control flow and interaction behavior. The script language is typically intended to have a simple, easy to use syntax.

1.3 Visual Programming

Shu (1988) defines visual programming as "the use of meaningful graphic representations in the process of programming" (p. 9). To us, however, visual programming is closer to Shu's definition of a visual programming language: "a language which uses some visual representations to accomplish what would otherwise have to be written in a traditional one-dimensional programming language" (p. 138). This definition is still quite broad, and indeed must be narrowed for clarification in the current context. In this paper, we use visual programming to mean the construction of a graph of interconnected icons which can be interpreted by an execution engine to perform a series of tasks.

In a "pure" visual programming system, all program tasks would be specified through an iconic interface. None of systems we know of that are usable for real applications yet reach this goal, and it is not clear that the goal is fully desirable. That is, some things are just easier to specify through textual commands or dialog boxes than through the manipulation of icons. Shu therefore analyses visual programming languages along three dimensions: their level, scope, and extent (pp. 139-141).

- Language level is "an inverse measure of the amount of detail that a user has to give to the computer in order to achieve the desired results." Higher levels visual programming

languages provide icons that representing high-level functions, that is, functions that perform compound and/or complex tasks such as presenting a named video sequence as opposed to low-level functions such as positioning a videodisc player to a specific track, turning the video signal on, playing to another track, and then turning the video signal off.

- Language scope describes "how much a language is capable of doing." Visual programming languages often display the quality of making simple tasks like displaying a video sequence truly trivial, but making standard programming tasks of very modest complexity, like extracting the first word of a student's response, difficult if not impossible. Achieving balance between these extremes if quite difficult without compromising the visual programming paradigm.

- Language extent for visual programming refers to "how much visual expressions are incorporated in the programming language." As mentioned above, we know of no systems that are purely visual, and we're not sure that a purely visual programming system would be desirable. However, some visual programming systems require an inordinate amount of textual programming, particularly for implementing conditional execution. Unfortunately, some such systems have only very low-level programming capabilities, making even modest flow-of-control constructs difficult to implement.

1.4 Object-Oriented vs. Iconic

Iconic authoring systems are often mistakenly considered to be object-oriented because some such systems call icons objects, and virtually all object-oriented systems use icons. To clarify matters: an icon is a graphical representation of a function, while an object is a programming entity made up of a type definition *plus* definitions of the operations that can be performed on objects of that type. In addition, object-oriented systems allow the definition of general properties of a class of objects plus specific properties of subclasses of that object.

For example, in an object-oriented authoring system one might define the general properties of a user input action and then distinguish the specific properties of mouse, keyboard, and audio input. Thus, while some iconic authoring systems are *built* using object-oriented technology, that is, they are implemented using an object-oriented language, their appearance to multimedia authors is iconic, not object-oriented. Iconic authoring systems use icons to represent fully contained functions that can be combined in various ways to create multimedia programs.

1.5 Authoring Process

The authoring process typically starts with a storyboard which lays out the general organization and content of the presentation. The storyboard evolves as the media are collected and organized; new ideas and refinements to the presentation are added as the presentation takes shape. The author/artist replays parts of the presentation during this refinement process.

The storyboard is informal and high-level. The emphasis is on facilitating the creative process by using sketches, screen grabs and other rapid input forms. The storyboard can be retained with the final presentation to show overall structure and content.

2. Review and Comparison

2.1 Package Selection and General Criteria

IconAuthor (AimTech, 1992) and *Authorware Professional* (Authorware, 1989) are both widely used and highly-rated commercial multimedia authoring tools. Each is available on multiple platforms and is suitable for developing applications for training or presentations. Both use the visual programming paradigm, but with notable differences. For these reasons, we selected these two packages for study.

We are primarily interested in evaluating and improving the user interface of multimedia authoring tools which employ visual programming techniques. We do not consider many issues that would be important in selecting one of these packages for actual use, such as integrated tools, performance, platform, or data interchange. Instead, we focus on two areas: the semantics of the icon set and the human factors of the iconic interface.

Icon set semantics concern identifying the meaning of the primitives and constructors defined by the visual programming language, including individual functionality and compositional methods. Programming language theory has several systems for developing formal semantics of a programming language. These techniques are used to develop systematic answers to questions related to functionality and correctness. Visual languages which closely follow procedural languages, or which can be translated into a procedural language, will borrow their semantics from the corresponding procedural model. *The practical consequence of semantic analysis is to answer questions such as what presentation and interaction sequences can be represented by the language, and whether all iconic compositions have a consistent interpretation.* Additionally, multimedia authoring tools include methods for temporal composition and synchronization; the correctness, expressiveness, and precision of these facilities is perhaps a unique aspect of multimedia authoring languages when compared to traditional programming languages.

In visual programming, the user manipulates a pictorial representation of a sequence of actions to achieve some larger function. Many factors influence the power and usability of such an interface, including the consistency and complexity of the icons, the graph organization, and the number of steps needed to perform editing operations. The domain of multimedia authoring, because of its visual orientation, adds the relation between the iconic graph and the visual aspects of the presentation. We are particularly interested in the relationship between the graph construction and editing activity and the multimedia authoring process.

2.2 IconAuthor

IconAuthor provides a visual programming editor in which the control icons closely correspond to typical programming language functions such as loop, if-then, and subroutine. The icons are single function, with several composition icons provided to build reusable collections of icons. Each icon's function is accessed through an associated content editor. The content editor for every icon has the same appearance, making it easy to use. There is a drag and drop window which contains the current graph; several graphs can be opened simultaneous. Editing functions are available through a set of pull-down menus; an icon ribbon provides quick access to frequently used functions. A set of integrated tools is included for text, graphics, animation, image

editing, and video control. These tools can be invoked from either the associated icon or the pull-down menus.

2.2.1 Function and Semantics

The icons are single function, with several composition icons provided to build reusable collections of icons. These are grouped in the following families:

- Flow: branches, if-then, loop, menu, module
- Input: keyboard input, mouse input
- Output: audio, structured graphics, erasure, graphics attributes
- Data: database and file access, system variables, built-in math and string functions
- Multimedia: audio, video card, video player
- Extensions: DDE and DDL interfaces, subroutine, a help facility, RS-232 control
- Custom: user defined icons

2.2.2 Human Factors

There are a large number of icons, but because the icons are primitive in function, they are easy to learn. The tool does not enforce any specific programming discipline, so it is quite possible for an author to create complex unstructured graphs. Because the icons are primitive in function, graphs tend to be large and hard to inspect. Although the tool provides techniques for managing the size of the graph, the author must still think at a relatively low-level of icon function during editing.

The graph area can be scaled, making it easier to navigate large graphs. Groups of icons can also be collapsed or expanded as desired, helping to control graph complexity. The icon ribbon makes a number of frequently used operations quite accessible. *IconAuthor* provides an extensive on-line help facility.

2.3 Authorware

Authorware is a visual programming system for Windows and Macintosh systems. It is based on PCD3, a project originally undertaken at Control Data Corporation. *Authorware* uses a flowchart visual programming paradigm. The author drags icons from an icon palette onto a flow line to specify the flow of control through the program. Each icon has a number of parameters that are set through dialog boxes, to which the author gains access by double clicking the icon. The beauty of *Authorware* is in the low number (seven) and apparent simplicity of its icons, although this simplicity can be deceptive due to the large number of options provided through dialog boxes. *Authorware* supports standard multimedia extensions, integrating them into the visual programming system with exceptional smoothness.

While *Authorware* provides a wide array of facilities, it is difficult to extend. The attractiveness of this system is in its high degree of attention to human factors and the consequent elegance of its implementation. It is easy to learn and a pleasure to use. Its main drawback is one shared by

all visual programming systems: when programs become complex, the difficulty in managing the icon network increases geometrically.

2.3.1 Function and Semantics

Authorware is based on a procedural model. Its seven icons are: display, erase, wait, calc, interaction, decision, map (a collection of other icons). Of these, the interaction and decision icons are the most deceptive, as they provide a huge amount of functionality through extensive use of options specified in dialog boxes. In addition, these two icons actually introduce sequences that tie together combinations of the other five icons in a structured manner. For example, the interaction icon can be used to accept mouse or keyboard input, analyze click locations, strings, and numbers, and provide conditional feedback for any number of alternatives. The decision icon can be used to control program flow in a simple if/then manner or through timed or numbered iterations.

The calc icon is also deceptive. It can indeed be used to set and retrieve values of variables and perform computations, but it also acts as the interface to other programs via DDE and DDL. We have seen some *Authorware* programs that are up to 30% calc icons, which somewhat defeats the purpose of the visual programming paradigm. The syntax of statements in calc icons is C-like, but unfortunately only 424 characters are allowed, making it difficult to write substantial routines.

Authorware contains a number of sophisticated options on icon functionality that can be difficult for the new user to grasp. For example, calculations can be "attached" to other icons without the need for calc icons. The functionality of an interaction icon includes a required display, which can be annoying when one wants to enter began an interaction sequence without changing the current display. The order in which interaction tests are carried is critical, and can cause interesting problems in some complex interaction sequences.

All in all, these semantic difficulties are not difficult to master; they just take time. To a programmer used to straightforward textual function calls, however, the specification of visual algorithms can be tedious. We believe this to be a characteristic of all production visual languages, and one that needs to be addressed through additional study.

2.3.2 Human Factors

Authorware is particularly strong in the area of human factors. With only seven icons, the system has a distinctly "approachable" feel, especially to novice users. It is an easy system to try things out in, as undos are trivial. The system has an exceptional degree of consistency and is very smoothly integrated with its underlying window system (either Windows or the Macintosh). The system designers have taken particular care to address novice computer users with such characteristics as allowing spaces in variables names.

The main problems we see in *Authorware* are:

- it forces a strict hierarchical structure (only one icon is allowed below interaction choices, although this icon may be a map icon)

- windows cannot be scrolled vertically and are therefore limited to a maximum of about nine icons along the main flow line (although any of these may be map icons which may be nested to any depth)
- calc fields only allow 424 characters and the grammar of if statements allows only one statement in the then and else clauses
- maps are not true subroutines (each time a map is inserted it is a copy of the code is inserted; therefore, changing a visual algorithm that is used repeatedly may require that change to be made in many places)
- no on-line help facility

2.4 Summary

Tables 1 and 2 summarize the comparison of these two tools for the categories of Functionality and Human Factors.

Table 1. Functionality and Representation Power

Measure	*IconAuthor*	*Authorware*
Icon Set Size	Large	Small
Icon Set Functions	Mixture of primitives and composition icons	Multi-function control icons
Icon Set Semantics	HLL-like; flowchart	Special
Icon Set Extensibility	Extensible	Not extensible
Hierarchy and Abstraction	Discretionary; Multiple techniques: composite, subroutine, module	Forced: non-scrollable windows; inability to expand interaction icons vertically; map icon
Family Style	Flowchart	Special
Semantics of Composition	Follows high-level language	1. Loop in interaction icon 2. Composition of sequencer and interaction
Relationship of Presentation Structure to Graph Structure	Low structural relation	Low structural relation

Table 2. Human Factors of Visual Language

Measure	*IconAuthor*	*Authorware*
Visual Consistency	High	Very high
Visual complexity and inspectability	Because icons are more primitive in function, complex graphs can result. This can be alleviated by use of modules and composites.	Low for individual windows, but can be high for deeply nested maps.
Graph size and relation to presentation function	If composites are used, graph size can be kept low	Average; many standard functions can be programmed using maps nested to only 1 or 2 levels
Use of color	Color used for icons and background; some customization available	Color not used in graph construction, but can be used extensively in displays
Text Labeling	Label icons	Label graph
Number of interaction steps needed to create something	Slightly high, because indirect access to smart text and graphics editors through icon content editors	Average, many choices provided in easy-to-use dialog boxes, especially for the interaction icon
Number of interaction steps needed to modify something	Slightly high, because of indirect access to certain editors	Average, with some types of interactions difficult to conceptualize, but we see this as a standard visual program problem, not a unique characteristic of *Authorware*
Debugging, in particular how debugging operations such as trace and breakpoint are integrated with the interface	Difficult to find currently executing icon; can selectively enable and disable icons	Good ability to jump to the icon controlling the current display, but no single step capability; excellent facility to insert stop and stop “flags” to control partial execution, but only one pair of flags can be inserted for any one run; can’t skip over or disable icons
Support for authoring process	Various icons for hierarchical design, and some predefined composite icons	Good, use of “models” as program templates for standard operations, also ability to insert empty map icons as “placeholders” that will be completed later

3. Discussion

Visual programming is still a relatively new field with many human factors issues yet unsolved. Basic to these issues are the level, scope, and extent metrics identified by Shu. While the visual programming paradigm is appealing to a wide range of users, it has drawbacks in a number of areas.

- All visual programming systems use icons. Icon design is easy when metaphors are clear, but very difficult for specifying constructs like iteration and recursion. None of the systems we have discussed allow users to define their own icons, and this can have a significant adverse effect on the visual readability of a visual program when most of its icons are subroutines (*Authorware* maps). Internationalization of icons is particularly difficult for abstract constructs.

- All visual programming systems also use menus. Menu layout for complex systems involves many decisions, the most important of which are what to name menu items and under which pulldown menu to place them. While there are some standards for common items, e.g., Undo, Cut, Copy, Paste, and Delete are typically the first five items on the Edit pulldown menu, application-specific items are often difficult to place. These problems are especially acute when developing for a cross-cultural audience.

- Visual program layout is typically either fixed, in which icons are laid out in a grid like *IconAuthor*, or free, in which icons may be placed anywhere in a visual program palette. Note that *Authorware* appears to have a freer layout than *IconAuthor*, but in truth it is the same, because given the program position in which one wants to place an icon to perform a certain function, there is only one place on the screen where that icon can be placed to achieve that sequence. Fixed layouts initially seem more restrictive, but users of free layouts often find that they spend a lot of time making their visual programs “look pretty,” and most often lay out their icons in a graphical style anyway.

- Icons are typically connected using lines. On some systems, these lines may are fixed, while on others they are movable. Fixed lines are fine when visual programs are laid out in a grid-like manner as described above, but can be troublesome in systems that allow free icon placement. Once again we have the trade-off of apparent rigidity versus enhanced visual control. When connections can be moved, there must first be an algorithm that determines their initial placement. Movement must then be constrained or the connections quickly degenerate into unintelligible “spaghetti” in which the connections are impossible to follow.

- One of the most difficult human factors issues in visual programming is the management of programs with many—hundreds, if not thousands—of icons. It is not clear to us that visual programs with thousands of icons have yet been built. As the number of icons grows, it becomes increasingly difficult not only to “see” the entire program, but to find specific routines that one is interested in copying, modifying, or correcting. *Authorware* allows authors to group icons into subroutines, and *IconAuthor* provides the ability to zoom in and out. These techniques are indeed very useful, but they do not solve the entire problem.

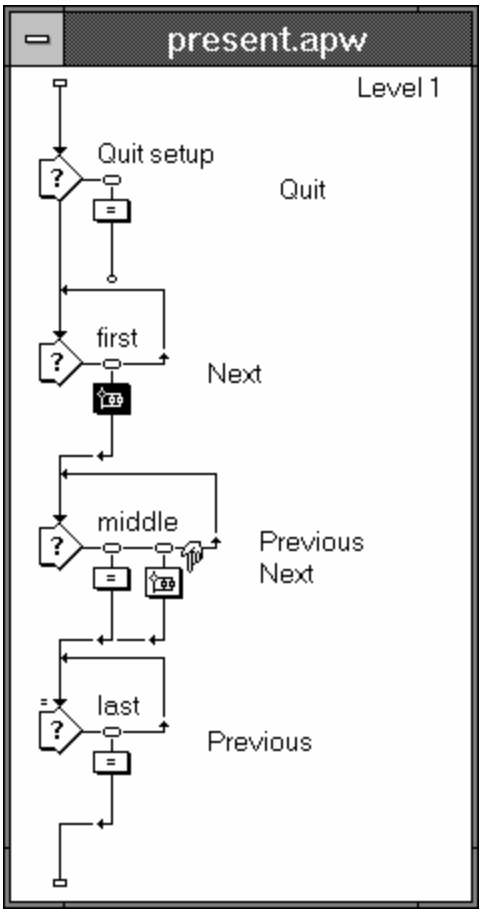
- Along the same lines, visual programs present numerous problems in the management of screen "real estate." Some systems allow the author to control icon sizes, but these can become too small to recognize. Window scrolling also helps address the issue. However, when one has numerous windows open with multiple visual programs and editing tools, these techniques can once again break down. Even when one iconifies windows to remove them from the immediate visual plane, one must still face the problem of finding the right icon to enlarge again.

- Finally, visual programs are particularly difficult to track during program execution so that they can be debugged. Some systems add animation to visual program execution, but this typically has a very high price in terms of system performance. Other systems allow the user to jump to the currently executing icon, but again this can be difficult if one doesn't know exactly where an error is occurring. That is, an error which shows up when a dialog is displayed may be caused by any one of numerous icons that precede it in the execution sequence.

4. Benchmarks

In this section we present an example benchmark for evaluating different multimedia authoring tools. Benchmarks that we have used to compare tools include:

- Sequential presentation
- Menu-based presentation
- Animated model
- Interactive hypertext and hypermedia



4.1 Graph Structure for a Sequential Presentation: Authorware Professional

The example is a simple program to present slides and allow the user to move forward and backward through the slides or quit at any time. The *Authorware* structure of such a program for three slides is shown at the right. The slide content is placed in the interaction icons (the arrow-like icons labeled with question marks in the figure). The first interaction, labeled "Quit setup," sets up a permanent Quit button that appears on all slides and which, when pressed, exits the presentation.

The interaction icon labeled "first" (highlighted in the figure) contains one push button labeled Next. When this button is pushed, control passes to the map icon directly under the "first" label, after which it exits the first interaction and enters the next one. In this case, the map icon is empty. (You need to have something for the interaction icon to do, and an empty map icon fills this syntactic need.)

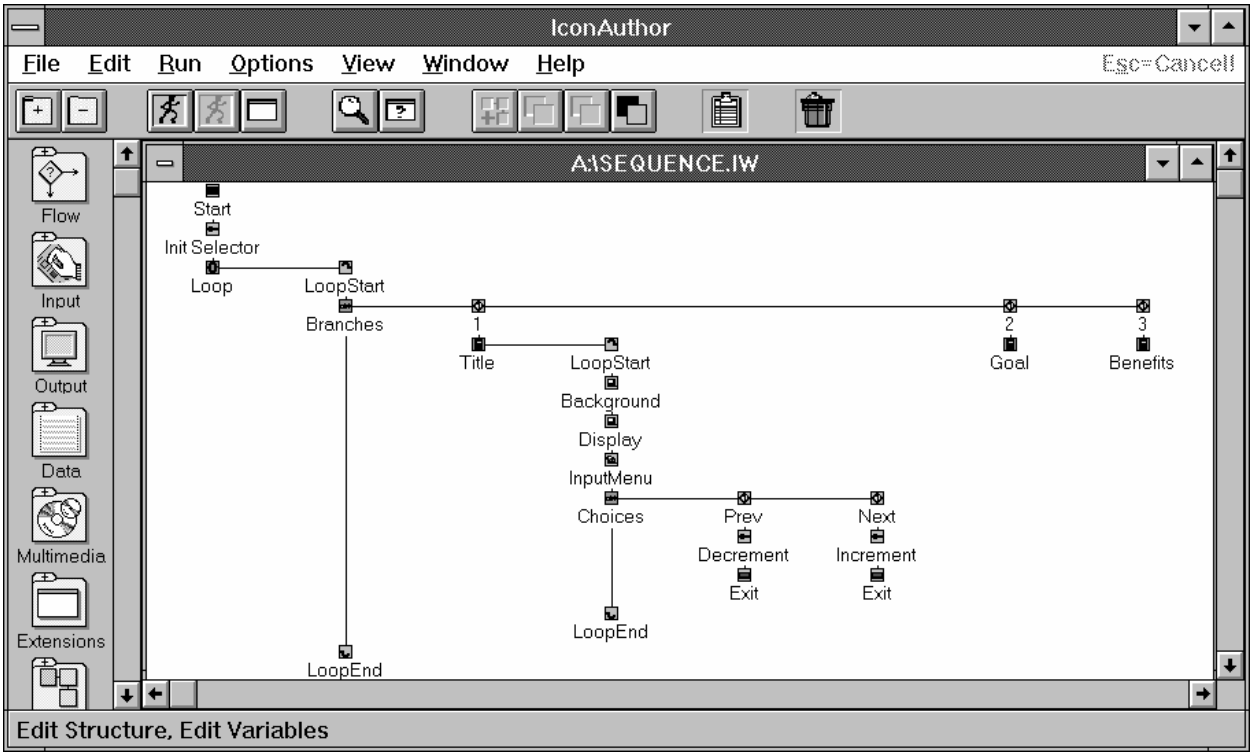
The icon labeled “middle” has two push buttons, labeled Previous and Next, respectively. The one labeled Next has an empty map icon and works just like that in the previous interaction. The one labeled Previous, however, has to explicitly jump to the previous icon, so a calc icon (the one labeled with an = sign) is used containing the statement:

```
GoTo(IconID@”first”)
```

This syntax is a bit cumbersome—and certainly not visual—especially since the name of the icon to branch to must be a string constant and cannot be a variable. However, *Authorware* is does update icon names embedded in calc icons if you happen to change the name of the icon. When we cut and paste icons to enlarge the slide show, however, we must open the calc icon and explicitly change the name of the previous interaction icon to go to. We don’t have to do this for the Next button, because its map icon is simply a place holder and the built-in interaction icon functionality automatically goes to the next icon when the interaction exits.

4.2 Graph Structure for Sequential Presentation: IconAuthor

The sequential presentation in *IconAuthor* shown below relies on an index variable to specify the screen to display. In this graph, successive screens are numbered 1, 2, 3, ...; at any point in the presentation, a branch forward is performed by incrementing the index variable and a branch backward is performed by decrementing the index variable. Each screen has an equivalent sub-graph structure, as shown for the first screen, where the graph has been expanded.



After the screen contents are presented, two choices are activated. Selecting a choice leads to the index variable being updated and control returns to the outermost loop. For presentations with many slides, the branch lists should be broken into segments to facilitate insertion and deletion. The mechanism for branching is awkward because it relies upon numeric labeling. Insertion and deletion of screens requires reediting of indices. Some type of symbolic branching support would be a significant improvement.

5. Summary and Conclusions

The visual programming paradigm therefore presents a number of fascinating human factors problems, but none of these seems strong enough to stem the tide of interest in this appealing technique. We believe that the best visual programming systems will be those that allow a high degree of user customization to address these problems, such as defining one's own composite building blocks with their own icons. In addition, these systems must allow users to break out to standard computer languages when textual approaches are more efficient. Like many programming tools, we believe that visual programming has great promise for multimedia authoring, but we do not want to be locked into using this paradigm when others are more efficient.

6. References

AimTech Corporation, 1991. *IconAuthor User Manual*. Nashua, NH.

Authorware, Inc., 1989. *Authorware Professional Manual*. Redwood City, CA.

Chang, S.-K. *Visual Languages: A Tutorial and a Survey*. IEEE Software. Jan. 1987. pp. 29-39.

Kieras, D.E. *Towards a Practical GOMS Model Methodology for User Interface Design*, in *Handbook of Human-Computer Interaction* (ed. Helander, M.), Elsevier, 1988, pp. 137-157.

Buford, J., Rutledge, J., and Heines, J. *Visual Programming Abstractions for Interactive Multimedia Presentation Authoring*. Proceedings of the 1992 IEEE Workshop on Visual Languages. 1992.

Shu, N.C., 1988. Visual Programming. Van Nostrand Reinhold, NY.