The Demonstrational Interfaces Project at CMU

Brad A. Myers,
Francesmary Modugno, Rich McDaniel, David Kosbie, Andrew Werth,
Robert C. Miller, John Pane, James Landay,
Jade Goldstein, and Matthew A. Goldberg

Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
http://www.cs.cmu.edu/~bydemo
bam@cs.cmu.edu

Abstract

The Demonstrational Interfaces Project at CMU has been investigating various aspects of demonstrational interfaces for the last eight years. During this time, we have created six interactive software building tools that use demonstrational techniques, as well as an architecture to support demonstrational programming in general. In addition, we have created a demonstrational Visual Shell (iconic interface to a file system, like the Macintosh Finder), a demonstrational text formatter, and a demonstrational charting tool. There are three fundamental research questions we explore through these tools: how to give the user appropriate feedback and control over inferencing, appropriate algorithms for inferencing, and which domains are appropriate for demonstrational techniques. This paper summarizes our activities, approach and lessons learned.

Introduction

In the User Interface Software Project and the Demonstrational Interface Project in the Human-Computer Interaction Institute at Carnegie Mellon University, we have created a variety of demonstrational, interactive programs. Like most demonstrational projects, the main goal is to allow non-programmers to achieve the effect of programming without needing to learn a textual programming language. We also wrote a number of general papers about Demonstrational Interfaces (Myers, 1992, Myers, 1993a, Myers, 1990a), to help popularize and explain the area. All of our demonstrational systems have used heuristics to try to automatically generalize from the user's examples, unlike systems such as SmallStar (Halbert, 1984) and KidSim (Smith, 1994). Some of our systems, including Pursuit and Marquise, take the conventional approach of recording a script of the user's actions, but most of our systems generalize from the *results* of a demonstration and so are not sensitive to the particular sequence of actions used to construct the examples.

This paper provides a summary of our various demonstrational systems, and then discusses our lessons learned.

Summary of Systems

As part of the large-scale User Interface Software Project, which developed Garnet (Myers, 1990b) and now Amulet (Myers, 1995), we created a number of interactive demonstrational tools. Another set of systems were created as part of the Demonstrational Interfaces Project. In chronological order, the systems are Lapidary, Jade, C32, Gilt, Tourmaline, Marquise, Pursuit, Katie, Gold, Silk, Amulet's command objects, and Gamut. Lapidary, C32, and Gilt are being distributed as part of the Garnet system, and can be retrieved from http://www.cs.cmu.edu/~garnet. The Amulet command objects are distributed as part of the Amulet system, and can be retrieved from http://www.cs.cmu.edu/~amulet. Development of Gold, Silk, Amulet, and Gamut is on-going. Lapidary, C32, Gilt and Jade are also described in the Garnet chapter of the PBD book (Myers, 1993b) and there are separate chapters in the book for Tourmaline, Katie and Pursuit.

Lapidary allows new widgets and new application-specific graphical objects to be created by drawing pictures and then specifying the behaviors using dialog boxes (Myers, 1989). The demonstrational part is that the objects are automatically generalized into prototypes which can be instantiated at run-time. Feedback in Lapidary includes a novel iconic representation for showing constraints, and dialog boxes for confirming inferences.

Jade automatically generates dialog boxes from a list of their contents (Vander Zanden, 1990). The primary demonstrational component is that some of the layout and design rules can be specified by example.

C32 is a spreadsheet interface that allows complex constraints to be specified (Myers, 1991a). The main demonstrational aspect of C32 is that when constraint formulas are copied from one place to another, C32 generalizes and transforms the constraint appropriately to its new context.

Gilt is an interface builder that provides widget layout. Two demonstrational additions were made to Gilt. First, Gilt will infer *graphical styles* from an example dialog box, including the placement of objects and the properties to use (Hashimoto, 1992). The placement was based on *graphical tab stops* which provide a direct way to manipu-

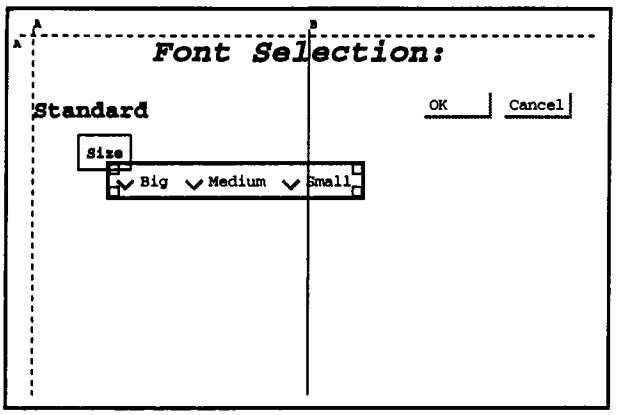

Figure 1: Gilt infers the positions of the horizontal and vertical "graphical tab stops" from the example graphics.

late the alignment (see Figure 1). After the styles were inferred from the example dialog box, future widgets placed near the inferred graphical tab stops would snap to them, and the new widgets would use the styles inferred from the examples. The second demonstrational component was a technique for eliminating "call back procedures" which are used to connect widgets to applications (Myers, 1991b). It turns out that many call-backs perform fairly mundane operations, such as making other widgets become active or inactive, changing the values of widgets, and calling built-in functions with specific arguments. Gilt allows these to be replaced by demonstrating the desired behavior. Gilt displays the inferred transformations as Lisp code, and allows the designer to edit the code or add additional examples.

Tourmaline is a text formatter that allows "macrostyles" to be specified by example (Myers, 1991c, Myers, 1993c, Werth, 1992). Tourmaline was built into Microsoft Word, and was the Masters thesis of Andrew Werth. Unlike the styles in conventional editors like Microsoft Word, Tourmaline allows a macrostyles to contain different formatting for different parts of a header, such as the title, author and author's affiliation. For example, the user could format an example header like:

Demonstrational Interfaces

Brad Myers
Carnegie Mellon University
bam@cs.cmu.edu

The user would then select the entire block of text, and define a macrostyle from it. Next, the user could select a different heading and apply the macrostyle to it, and Tourmaline would search for the various parts (title, author, e-mail address) and apply the appropriate style to each. If a heading had two authors, Tourmaline would know how to spread them out appropriately. Tourmaline uses heuristics to distinguish the role and formatting of the parts and provides feedback through dialog boxes.

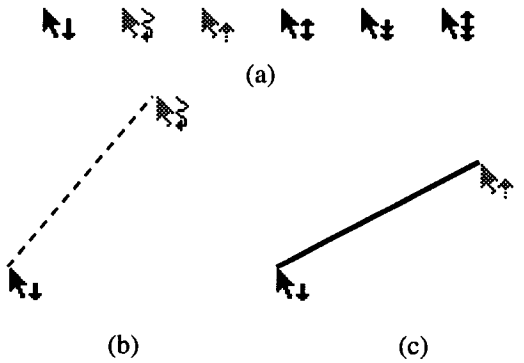

Figure 2: (a) The icons that show where the mouse was pressed, moved to, released, clicked (pressed and released in the same place), double-clicked, and double clicked and released. (b) In stimulus mode, the designer pressed the mouse down and moved, and then in response mode, drew a dotted line as the interim feedback from one icon to the other. (c) Going back to stimulus mode, the designer released the mouse button, and in response mode, deleted the dotted line and drew a solid line.

Marquise is used to create graphical editors by demonstration. It supports demonstrating how the low-level "rubber-band feedback" looks, how selection handles are drawn and how they behave, how objects are created, and how palettes control what objects are created and how they look. As the user demonstrates events that cause an action (the "stimulus"), Marquise drops icons on the window showing where the events occurred, as shown in Figure 2. The user can then refer to these icons directly when demonstrating the response. This technique was subsequently used in Grizzly Bear (Frank, 1995), and will be substantially generalized in our new Gamut system. Marquise also invented a new form of feedback for inferences where phrases in textual sentences serve as buttons that pop up alternative options (see Figure 3).

Pursuit was the PhD thesis of Francesmary Modugno, and is a visual shell (an iconic interface to a file system, like the Macintosh finder) (Modugno, 1995, Modugno, 1993, Modugno, 1994a, Modugno, 1994b). As the user demonstrates a program by executing example commands, Pursuit builds a visual language representation of operations and inferences. The visual language is based on the "comic strip metaphor." The panels show the relevant data, and changes from one panel to the next represent the operation (see Figure 4). This language is innovative because, unlike most other textual and visual languages, it represents the data and leaves the operations implicit. The

language provides a single medium for verifying and correcting inferences, for reviewing completed programs later, and for editing programs. Formal human factors experiments showed that the language and system were successful and usable. Pursuit two main forms of generalization were to infer set descriptors and loops. If the user selected a number of files and then did an operation on them, Pursuit would try to define a set that matched the selected files and none of the un-selected files. For example, the user might have selected all the files of a certain type, or all edited after a certain data. If the user performed the same operations twice, Pursuit would notice this and try to form a loop. The feedback for these inferences was the visual language shown in Figure 4, and the user could directly edit the generated script if Pursuit made an error.

Katie is the PhD thesis of David Kosbie, and is an architecture for supporting script-based demonstrational programs (Kosbie, 1994, Kosbie, 1993, Kosbie, 1996). An important innovation in Katie is the support for "aggregate events" which means that user's actions are recorded at multiple levels: the low level event stream as well as the higher-level actions resulting from these events. Katie investigates how users might decide at what level scripts containing these aggregate events might be played back. Katie inspired the novel hierarchical command architecture now used in Amulet (see below).

Gold allows custom business charts to be specified by demonstration (Myers, 1994). Gold supports column charts, stacked column charts, line charts, pie charts, and many forms of scatter charts. The user draws a few examples of the charting elements, such as rectangles or circles, and the system generalizes to make elements for all of the data in a spreadsheet. Graphical "link boxes" show the inferred association of the graphics to the spreadsheet. Gold handles various forms of bar charts, stacked bar charts, scatter plots, line charts, x-y plots, and pie charts. Gold also supports highlighting a special value of the data series. The user can change the color or other property of an object, select it, and declare it to be a "special marker," and Gold will try to determine why the item is marked. Alternatively, the user can draw and select extra items to be used as special markers, such as an arrow and a label might be drawn next to the important item.

Silk is the PhD research of James Landay, and it allows graphic designers to sketch an interface with a pen on a computer tablet (Landay, 1995, Landay, 1996). Silk recognizes the widgets as they are drawn and allows the interface to be exercised for testing. Storyboards can be sketched to show the temporal behavior of the interface. The inferred type of objects are shown by highlighting the name in a button panel, and the user can cycle to the next guess or explicitly select the correct choice.

Amulet is a new user interface development environment that contains a new architecture for command objects. This promises to provide a powerful framework for supporting script-based demonstrational interfaces (Myers, 1996). This is inspired by the hierarchical events of Katie, described above. When input arrives or a widget is operated by the user, instead of invoking a call-back procedure as in most other toolkits, Amulet allocates a command object and calls its DO method. Unlike previous uses of command objects, Amulet organizes the commands into a hierarchy, so that low-level operations like dragging or selection invoke low-level commands, which in turn might invoke widget-level commands, which invoke high-level, application-specific commands, and so on. The top-level commands correspond to semantic actions of the program. As commands are executed, they can be transcribed for later undoing, repeating or analysis. We feel that this will

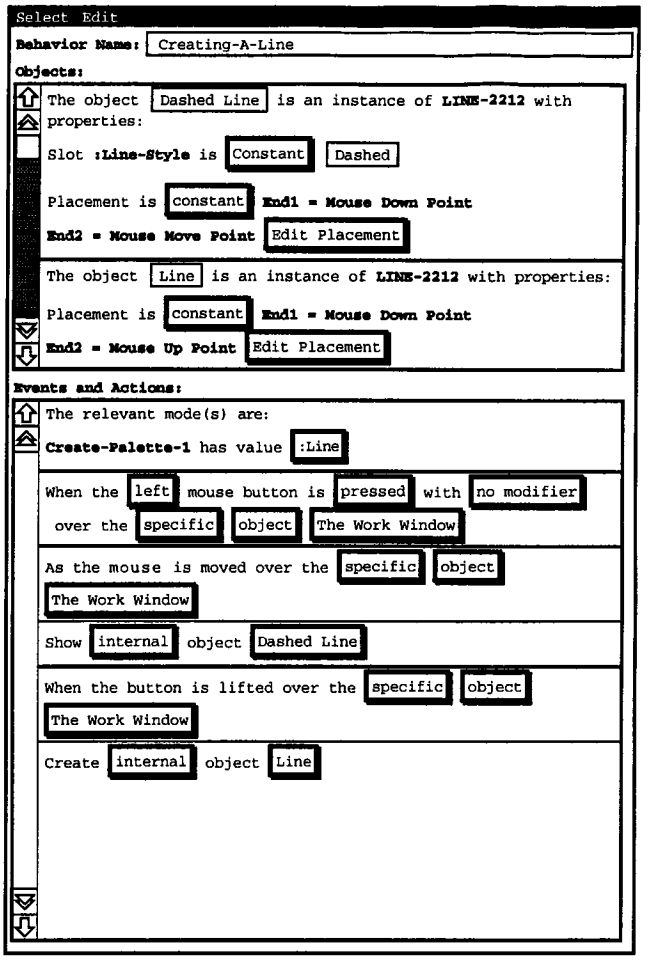

Figure 3:
The feedback window used by Marquise for inferred behaviors. At the top is a pull-down menu of commands, then the name of the behavior, then the objects that participate in the behavior, and finally the events and actions. Pushing on the buttons displays a popup window of the other possible choices. Changing the option at the beginning of a "sentence" will change the options available for the rest of the sentence. An entire section of the window can be selected and cut, copied, etc.

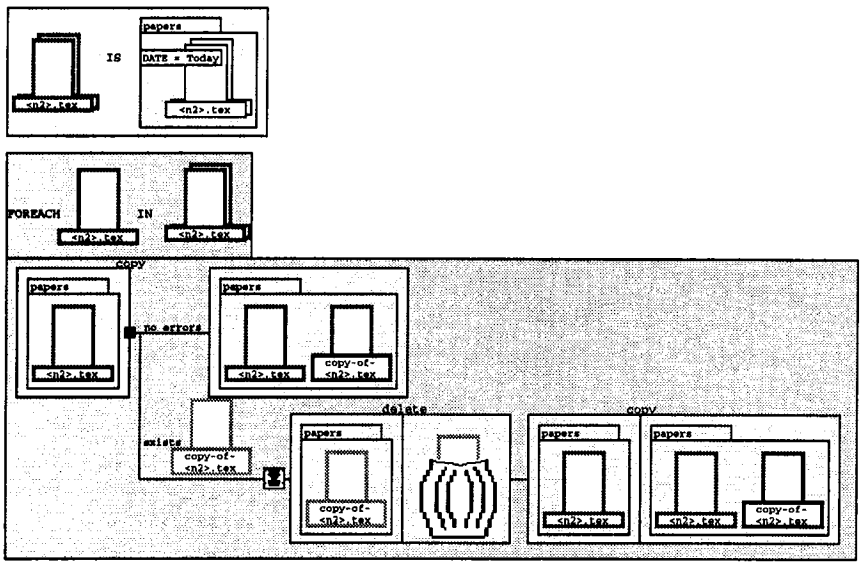

Figure 4: A complex program in the comic-strip language of Pursuit containing an explicit loop and a branch. The program copies each .tex file in the papers folder. The initial copy operation either succeeds (the upper branch) or fails (the lower branch). This conditional is depicted graphically by the branch (i.e., the little black square) and predicates after the first panel for the copy operation. Pursuit generated this program as the user demonstrated the actions on two actual file objects: one in which the copy executed successfully, and one in which the output file already existed in the papers folder.

better support demonstrational interfaces since both high-level and low-level information is available for matching. Thus, if the user executes a command with a menu one time and the same command from a keyboard accelerator, they can still be matched when searching for loops. Furthermore, the commands explicitly encode the parameters and results of the operations, so they can be investigated and generalized.

Gamut is the PhD research of Rich McDaniel, and is our newest demonstrational system. Gamut will allow non-programmers to construct complete games and educational software by demonstration. The main focus of Gamut is on new interaction techniques and metaphors to allow the system to infer the complex rules that control real games. We expect that Gamut will use Amulet's command objects. Gamut is described in another presentation at this workshop (McDaniel, 1996).

Lessons Learned

In creating all of these systems, we are researching three major questions: what are useful interaction techniques for specifying the demonstration and for providing feedback about what was inferred, what are the appropriate algorithms for inferencing, and which domains are appropriate for demonstrational techniques.

Feedback and Control

The most important question is how the users will control, understand, and correct the inferences made by the system. Any system that generalizes from examples will occasionally guess wrong, and it is important that users know what the system is doing so they will feel comfortable and in control. This is related to a number of issues in Artificial Intelligence systems, including confirmation dialogs in speech systems and explaining the reasoning in knowledge based systems. Furthermore, interaction techniques can help guide the inferencing engine to make more successful guesses.

In the early Peridot system (Myers, 1988), I used a conventional question-and-answer dialog to confirm inferences. Users found this disruptive and tended to answer "yes" without thinking. In Lapidary and Gilt interface builders and in the Tourmaline text formatting system, we used dialog boxes to confirm inferences, which seemed more successful. In Lapidary, a novel dialog box allowed the user to specify graphical constraints on the example objects, which would then be generalized to the prototypes. Different mouse buttons were used to select the "primary" and "secondary" objects to define relationships. In Gilt, the graphical tabs were displayed as design-time guide objects to concretely show the inferred alignments of objects. The user could then adjust and create tabs explicitly.

In Marquise, we invented a novel form of natural language feedback, where the user could click on phrases representing different parts of the inference to get a pop-up menu of alternatives (Figure 3). Experience showed that it was dif-

ficult to construct meaningful sentences with the appropriate options. Icons in Marquise representing the cursor positions and events are dropped onto the screen, so the user can make reference to these when drawing objects (Figure 2). When an object is next to one of these icons, the system give priority to object descriptions related to the cursor positions. This was very successful.

In the Gold custom charting system, text-input boxes appear near charting elements to show the inferred relationships. This made it easy to change the correspondences. Silk uses color to show the inferred groupings of objects, and displays the inferred types of objects in a menu. The user can cycle to the next best guess, or explicitly pick the desired type from the menu.

Pursuit incorporates our most successful and novel mechanism for feedback. Despite skepticism from other researchers that its graphical programming language would be usable, formal user studies showed that nonprogrammers could create fairly complex programs, and that the visual language was more effective than an equivalent textual language for generating programs.

In Gamut, we will be exploring a number of new interaction techniques to help guide the system's inferences, and to provide feedback to the user. This will be necessary since Gamut needs to make more sophisticated inferences than any previous demonstrational system.

The general lesson seems to be that graphical presentations work better than textual ones, and that users prefer a passive presentation of the inferences over one that interrupts with questions. For example, the graphical icons used for the constraints in Lapidary, the position of the mouse events in Peridot and Marquise, and the graphical language in Pursuit were more successful than the various question-and-answer techniques. Having concrete, manipulable results of the inferences, such as the tabs in Gilt and the visual language of Pursuit, is critical to allowing the user to correct and edit the resulting programs. Innovative interaction techniques can also guide the inferencing engines towards more successful guesses.

Inferencing Algorithms

The second fundamental research question we address is what are appropriate representations and inferencing algorithms. For a system to be acceptable, it must guess right most of the time. Most of our systems, including all of our early ones, used straightforward rule-based techniques and pattern matching that were empirically tuned to give acceptable performance. These resulted in highly predictable user interfaces, and this technique has been adopted by most other demonstrational systems. In Gamut, we are exploring a more elaborate algorithm to provide powerful inferences that can take into account the user's hints.

All of these algorithms use models of the domain that underlie applications. For example, in Peridot and Lapidary, we used constraint and behavior models, in the Tourmaline text formatter, we used a model of the most common forms of section headers, in Gold, we have a model of the most common business charts, in Pursuit, we used a model of the common operations in a visual shell, and in Gamut we will use a model of the common properties of board games. These models constrain the inferences and significantly increase their accuracy.

Domains

The third research question is to which domains can demonstrational interfaces be successfully applied. We have identified useful aspects of user interface construction, text formatting, business charting, file manipulation in a Visual Shell, and educational game construction that are appropriate for being demonstrational. They share the properties that the natural way a person would describe the problem to another person is by drawing examples, and that domain knowledge can be used to narrow the range of possibilities for generalizing from the examples. Each domain also illuminates new issues for feedback and representations.

In the future, we will be working to further develop demonstrational interfaces in new areas. In the Gamut tool, we will significantly expand the range of what can be created by demonstration, by inventing new interaction techniques, metaphors and algorithms. We will also expand the work on creating custom business charts and data visualizations by demonstration. Other areas we will look into include computer-aided manufacturing and the world-wide web.

Conclusions

The Demonstrational Interfaces Project has created many interesting systems in different domains. We believe that there is a great potential for demonstrational interfaces to be a "step beyond direct manipulation" (Myers, 1992) and we look forward to collaborating with AI researchers to incorporate more elaborate and accurate inferencing algorithms.

Acknowledgements

The Demonstrational Interfaces Project has been primarily funded by NSF under grants IRI-9319969 and IRI-9020089, and by the Hertz Foundation. The User Interface Software Project has been primarily funded by NCCOSC under Contract No. N66001-94-C-6037, ARPA Order No. B326, and by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

Martin R. Frank. (1995). *Model-Based User Interface Design by Demonstration and by Interview.* Doctoral dissertation, College of Computing, Georgia Institute of Technology. In progress.

Daniel C. Halbert. (1984). *Programming by Example.* Doctoral dissertation, Computer Science Division, Dept. of EE&CS, University of California. Also: Xerox Office Systems Division, Systems Development Department, TR OSD-T8402, December, 1984.

Osamu Hashimoto and Brad A. Myers. (November 1992). Graphical Styles for Building User Interfaces By Demonstration. *ACM SIGGRAPH Symposium on User Interface Software and Technology.* Monterey, CA: Proceedings UIST'92.

David S. Kosbie and Brad A. Myers. (1993). A System-Wide Macro Facility Based on Aggregate Events: A Proposal. In Allen Cypher (Ed.), *Watch What I Do: Programming by Demonstration.* Cambridge, MA: MIT Press.

David S. Kosbie and Brad A. Myers. (1994). Extending Programming By Demonstration With Hierarchical Event Histories. In Brad Blumenthal, Juri Gornostaev and Claus Unger (Ed.), *Human-Computer Interaction: 4th International Conference EWHCI'94, Lecture Notes in Computer Science, Vol. 876,.* Berlin: Springer-Verlag.

David Kosbie. (1996). *Hierarchical Event Histories in Graphical User Interfaces.* Doctoral dissertation, Computer Science Department, Carnegie Mellon University. In progress.

James Landay and Brad A. Myers. (May 1995). Interactive Sketching for the Early Stages of User Interface Design. *Human Factors in Computing Systems.* Denver, CO: Proceedings SIGCHI'95.

James Landay. (1996). *Interactive Sketching for the Early Stages of User Interface Design.* Doctoral dissertation, Computer Science Department, Carnegie Mellon University. In progress.

Richard McDaniel. (March 1996). Using Better Communication To Improve Programming-by-Demonstration. *AAAI'96 Spring Symposium: Aquisition, Learning, and Demonstration.* Boulder, CO: AAAI.

Francesmary Modugno and Brad A. Myers. (1993). Graphical Representation and Feedback in a PBD System. In Allen Cypher (Ed.), *Watch What I Do: Programming by Demonstration.* Cambridge, MA: MIT Press.

Francesmary Modugno and Brad A. Myers. (October 1994). A State-Based Visual Language for a Demonstrational Visual Shell. *1994 IEEE Workshop on Visual Languages.* St. Louis, MO: IEEE Computer Society.

Francesmary Modugno, T.R.G. Green and Brad A. Myers. (August 1994). Visual Programming in a Visual Domain: A Case Study of Cognitive Dimension. *Proceedings of Human-Computer Interaction '94, People and Computers IX.* Glasgow, Scotland.

Francesmary Modugno. (1995). *Extending End-User Programming in a Visual Shell with Programming by Demonstration and Graphical Language Techniques.* Doctoral dissertation, Computer Science Department, Carnegie Mellon University. Computer Science Technical Report CMU-CS-95-130.

Brad A. Myers. (1988). *Creating User Interfaces by Demonstration.* Boston: Academic Press.

Brad A. Myers, Brad Vander Zanden, and Roger B. Dannenberg. (November 1989). Creating Graphical Interactive Application Objects by Demonstration. *ACM SIGGRAPH Symposium on User Interface Software and Technology.* Williamsburg, VA: Proceedings UIST'89.

Brad A. Myers. (October 1990). Invisible Programming. *1990 IEEE Workshop on Visual Languages.* Chicago, Ill: IEEE Computer Society.

Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal. (November 1990). Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. *IEEE Computer, 23*(11), 71-85.

Brad A. Myers. (April 1991). Graphical Techniques in a Spreadsheet for Specifying User Interfaces. *Human Factors in Computing Systems.* New Orleans, LA: Proceedings SIGCHI'91.

Brad A. Myers. (November 1991). Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs. *ACM SIGGRAPH Symposium on User Interface Software and Technology.* Hilton Head, SC: Proceedings UIST'91.

Brad A. Myers. (Apr 1991). Text Formatting by Demonstration. *Human Factors in Computing Systems.* N.O., LA: Proceedings SIGCHI'91.

Brad A. Myers. (August 1992). Demonstrational Interfaces: A Step Beyond Direct Manipulation. *IEEE Computer, 25*(8), 61-73.

Brad A. Myers. (1993). Demonstrational Interfaces: A Step Beyond Direct Manipulation. In Allen Cypher (Ed.), *Watch What I Do: Programming by Demonstration.* Cambridge, MA: MIT Press.

Brad A. Myers. (1993). Garnet: Uses of Demonstrational Techniques. In Allen Cypher (Ed.), *Watch What I Do: Programming by Demonstration.* Cambridge, MA: MIT Press.

Brad A. Myers. (1993). Tourmaline: Text Formatting by Demonstration. In Allen Cypher (Ed.), *Watch What I Do: Programming by Demonstration.* Cambridge, MA: MIT Press.

Brad A. Myers, Jade Goldstein, and Matthew A. Goldberg. (April 1994). Creating Charts by Demonstration. *Human Factors in Computing Systems.* Boston, MA: Proceedings SIGCHI'94.

Brad A. Myers, Rich McDaniel, Alan Ferrency, Andy Mickish, Alex Klimovitski, and Amy McGovern. (June 1995). *The Amulet Reference Manuals* (Tech. Rep.) CMU-CS-95-166. Carnegie Mellon University Computer Science Department. also Human Computer Interaction Institute CMU-HCII-95-102. WWW = http://www.cs.cmu.edu/~amulet.

Brad A. Myers and David Kosbie. (April 1996). Reusable Hierarchical Command Objects. *Human Factors in Computing Systems.* Vancouver, BC, Canada: Proceedings SIGCHI'96.

David Canfield Smith, Allen Cypher and Jim Spohrer. (July 1994). KidSim: Programming Agents Without a Programming Language. *Communications of the ACM, 37*(7), 54-67.

Brad Vander Zanden and Brad A. Myers. (April 1990). Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces. *Human Factors in Computing Systems.* Seattle, WA: Proceedings SIGCHI'90.

Andrew J. Werth. (October 1992). *Tourmaline: Formatting Document Headings by Example.* Master's thesis, Information Networking Institute, Carnegie Mellon University.