Chapter 1

COMPUTER-AIDED DESIGN OF USER INTERFACES BY EXAMPLE

Henry Lieberman
Media Laboratory, Massachusetts Institute of Technology (MIT), 20 Ames St. 320 G
Cambridge, MA 02139 (USA)
E-mail: lieber@media.mit.edu – URL: http://lieber.www.media.mit.edu/people/lieber/
Tel.: +1-617-253-0315 – Fax: +1-617-253-6215

Abstract: A promising approach to Computer-Aided Design of User Interfaces (CADUI) is *Programming by Example*, where an interface designer demonstrates the behavior of an interface by presenting concrete examples and demonstrating how the system should behave on those examples. It lets the user interface designer "play end-user", simulating what an end-user would see and do. A software agent records the steps of the user interface and generalizes a program that can be used in analogous situations in the future. The popular genre of so-called Interface Builders can be seen as a "poor-man's" Programming by Example. It is now time to extend such systems so that behavior as well as appearance can be specified by example.

Key words: Programming by example, programming by demonstration, computer-aided design, interface builders, machine learning, software agents.

1. CAD/CAM AND USER INTERFACE DESIGN

The goal of Computer-aided Design of User Interfaces (CADUI) is to enlist the computer to aid an interface designer in constructing an interactive interface to be eventually used by an end-user. This point of view is by analogy to Computer-Aided Design and Computer-Aided Manufacturing (CAD or CAD/CAM). CAD traditionally refers to tools to visualize, describe, edit and test manufactured artifacts, which are now an indispensable part of all manufacturing and production processes.

Computer-Aided Design systems are successful in manufacturing because they replace the specifications of numerical parameters, a task difficult

for humans to perform reliably, with the visualization and editing of visual representations of virtual objects, which is much more natural for humans to perform. Computer-Aided Design systems can be successful in user interface design because they replace specification of programming language constructs, which are difficult for humans to perform reliably, with the visualization and editing of visual representations of virtual interfaces, which are more natural for humans to perform.

While manufacturing CAD systems are oriented towards describing the size and shape (and perhaps associated attributes such as materials or cost) of products, user interface CAD must not only describe the appearance and form of the user interface, but also the behavior of the interface. If, for a given interface, each interface element can have its own independent behavior, then a user interface often can be defined simply by combining and connecting pre-defined behavioral elements. But if there are interactions between the elements, if there is state to be maintained, or particular actions to be taken, then some other method for describing the behavior must be found. The usual approach is for the interface designer to write code in a traditional programming language, like C or Java, or text in a high-level interface description language.

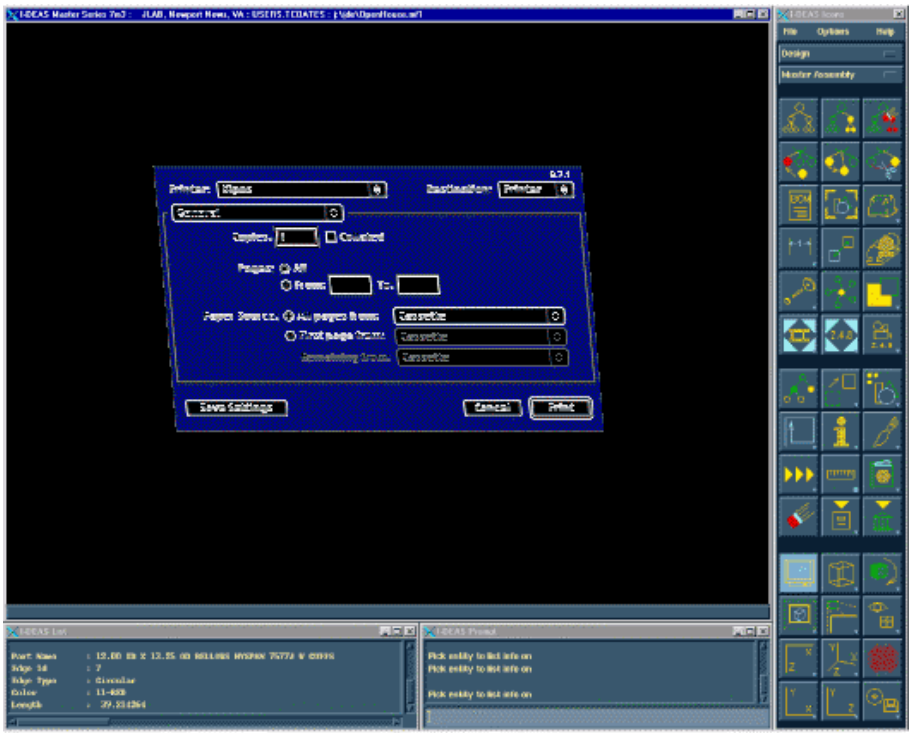

Figure 1. Can we design user interfaces the way we design CAD artifacts?

But writing and debugging in these languages is a difficult cognitive task for most people. To make the CAD analogy complete, it would be better if we could specify dynamic behavior by visualizing and editing virtual objects visually represented on the screen than by writing textual specifications, no matter how high level the language.

2. PROGRAMMING BY EXAMPLE

It is this goal which is the objective of *Programming by Example* (PBE, sometimes also called Programming by Demonstration). With Programming by Example, the user interface designer gets to "play user". He or she takes on the role of operating the user interface in the same manner as the intended end-user would, interacting with the on-screen interface components to demonstrate concrete examples of how to use the interface. The PBE system records the steps of the interaction, and synthesizes a program, in a similar manner to the way many interfaces contain so-called "macro" facilities that can record and playback interface steps. But in contrast to conventional macros, PBE systems have the ability to *generalize* programs, by using machine learning techniques to replace constants with variables, objects and actions with more general descriptions. The generalized program can then be used in situations analogous to, but not exactly the same as, those upon which it was taught. Programming by Example takes advantage of people's natural tendency to learn by example, and to teach by example.

Programming by Example systems have a long history, but PBE remains a minority viewpoint, relative to mainstream computer science. The PBE vision is quite radical, involving heuristics, machine learning, and agent approaches that conventional interface designers are not accustomed to. Over the years, a dedicated group of researchers has proved the feasibility of Programming by Example systems in an astonishingly wide variety of interesting domains as diverse as text editing, graphical editing, Web browsing and authoring, geographic information systems, animation, video games, educational applications, traditional CAD-CAM, and others.

There are two seminal collections of works in this field. The first, Allen Cypher's *Watch What I Do* [4], collects the initial works in this field dating back to David Canfield Smith's pioneering 1975 Pygmalion. It also includes a wealth of background material on this topic, such as a glossary, chronology, test suite and several survey articles. My recent book, *Your Wish is My Command* [12] brings together most of the important work in this field to date, detailing examples of over 17 different systems in the above-mentioned application areas, several of which are now commercially available. It incorporates a number of contributions from the international community of researchers in this topic.

3. PROGRAMMING BY EXAMPLE AND "INTERFACE BUILDERS"

Perhaps the closest conventional interface design tools to Programming by Example are the so-called "Interface Builders" – graphical editors that allow the interface designer to draw the position and size of so-called "widgets" – interactive interface elements such as menus, buttons, icons, text entry fields, etc. Interface builders trace their history to the early direct-manipulation interfaces, and were introduced in their modern form by Buxton [3]. More modern versions include the Microsoft Visual Basic editor, Apple's Hypercard and FaceSpan, or Macintosh Common Lisp's IFT Interface Tools. Animation editors used for interface "mock-ups" such as Macromind Director and Toolbook provide even more capabilities for including interactive elements and time-dependent behavior into interfaces. What's great about these systems is that they let you construct concrete examples of interfaces so that you can get immediate visual feedback as to the appearance to the end-user of the interface. They are the "poor man's" Programming by Example for interface design.

But all these tools break down at a certain level of complexity of the interface. It is usually not possible in such editors to allow the interface to take an arbitrary action in *response* to a user's selection of a button or menu item, except to invoke some already-defined function. The definition of such a function needs to be done in a conventional programming language, or, in many of the modern interface builder environments, in a so-called "scripting language". Visual Basic, Hypertalk, or Macromind's Lingo (which has a graphical representation) can also be considered scripting languages. Scripting languages are usually simplified programming languages that are deemed to be simpler for non-expert users to program, but they are typically quite limited in their ability to define new objects and new behavior, and most are quite poorly designed from the programming language standpoint. Worse, the interaction with the programming language is completely divorced from the interaction with the interface elements.

The approach of Programming by Example to interface design is to encourage the designer to switch roles, first casting the designer in the role of the end-user, where the designer simulates the actions that the end-user will eventually take in the interface. Then, in response, the interface designer simulates the role of how the interface will respond to the user's actions. The PBE system then records and generalizes the program. David Wolber refers to this as *stimulus-response PBE* [20]. An important aspect of stimulus-response PBE is that it generalizes on *time* as well as on example objects; the user demonstrates *when* to do something as well as *what* to do.

The first attempt to turn an interface builder into a true Programming by

Example system was Brad Myers' Peridot [15]. Peridot featured a "simulated mouse" that was used to demonstrate user actions so that it was not confused with the ordinary mouse used to interact with Peridot itself. Peridot used inference to generalize geometric relations in the interface such as when objects line up, replaced constants with variables in situations such as looping through a displayed list of choices and generated some simple conditionals.

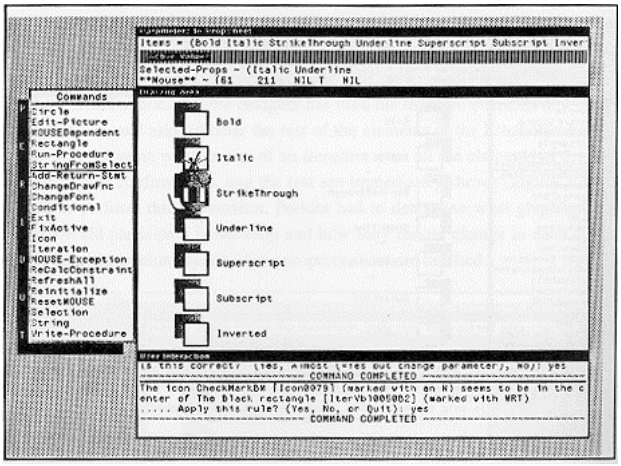

Figure 1. Peridot.

Myers and his group at Carnegie-Mellon University produced a long series of Programming by Example systems that operated in the Interface Builder domain. These are too numerous to describe in detail, but are surveyed in [14] and the papers referenced there. Notable among them were Lapidary; Jade, which created dialog boxes by example; Gilt, which permitted "callbacks" from interface elements; and, perhaps the most sophisticated, Marquise [16]. Several of these systems provided many ways for the user to edit properties of the generalizations constructed by the system. Elements of all these systems were incorporated in the general purpose User-Interface Management Systems produced by Myers and his group, first Garnet [15], then its successor Amulet.

Though not strictly an Interface Builder itself, my general purpose Programming by Example system, Tinker [9] used some Interface Builder-like techniques [interactive construction and placement of interface elements] and was used to construct interactive interfaces such as a spreadsheet-like interface [8] and a video game [9]. Tinker was perhaps the most procedurally general Programming by Example system, allowing functional abstraction, subroutines, and conditionals. Conditionals were defined from presenting

multiple examples, one example for each branch of the conditional. Tinker did require the knowledge of the programming language, and it required more explicit programming steps than many of the other Programming by Example systems, but the interface designer got more procedural power as a result. The interface designer typed in snippets of code, which were immediately executed, and the resulting objects, either code objects or interface objects, were then available as examples to be used in subsequent steps of the demonstration. Generalizations were propagated through the tree of dependencies in the code.

Programming by Rehearsal [5] allowed constructing interfaces by using the metaphor that the interface elements were "performance" in the interface "performance". Designers could "audition" the performers to learn their capabilities. Hidden variables were placed "backstage". It was used by teachers and kids for educational applications.

4. COMPUTER-AIDED DESIGN OF ANIMATED APPLICATIONS BY EXAMPLE

For applications that make more intensive of time-dependent and animated interfaces, such as computer games, some Programming by Example systems go beyond Interface Builders that define standard menu-and-icon interfaces to provide extensive facilities for defining dynamic, autonomously acting objects and demonstrating transformations. Still mainly in the Interface Builder paradigm, David Wolber's Pavlov system [20] takes the scripting and timeline metaphor appearing in animation interface systems such as Macromind Director, and giving them a Programming by Example component to demonstrate the dynamic behavior of individual objects in the interface and their interaction. Wolber's article has a direct comparison of his system with the more conventional Director.

David Canfield Smith, and Allen Cypher's Stagecast Creator [18] and Alex Repenning and Corinna Perrone-Smith's AgentSheets [17] use a rule-based metaphor. Rules are defined by constructing before-and-after examples that specify the conditions under which a rule is to fire, and the action to take as a result, in terms of how it transforms the before state to the after state. The principal application is for children to build computer games. Both use a grid world and finite-state transitions to structure the domain to model the movable game characters and (relatively) fixed scenery.

A radically different approach is taken by Ken Kahn's Toontalk [6]. In Toontalk, the programming environment itself is an animated video game. Programming language constructs are animated video characters. Robot characters represent user-defined programs whose thought-bubbles contain

the code, animated actions demonstrated by example. A variety of physical-world metaphors, cities, houses and trucks, birds and nests, etc. are used to explain the behavior of the program constructs, which are based on an underlying model of concurrent logic programming.

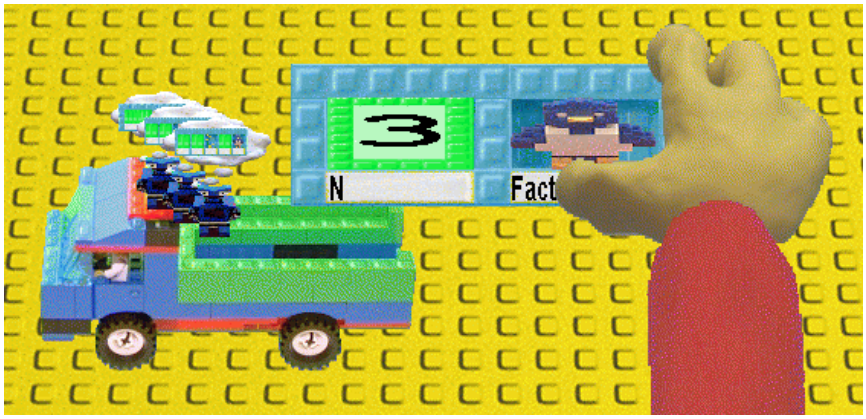

Figure 3. Toontalk.

5. COMPUTER-AIDED DESIGN OF WEB APPLICATIONS BY EXAMPLE

These days, many Web pages themselves can be considered user interfaces. Although a Web browser is often regarded as a specific application, Web pages that have embedded menus or forms, Javascript, Java applets and the like can be regarded as independent user interfaces, and Web page authoring amounts to interactive interface design. We are beginning to see Programming by Example applied in this domain as well. Web pages are constructed by operations of browsing, cutting and pasting from other Web pages, as well as giving advice to the system about how to interpret the format of Web pages.

Two examples are Atsushi Sugiura's Internet Scrapbook [19] and Matthias Bauer, Dietmar Dengler and Gabriele Paul's Trias [1]. Trias is especially interesting as a Programming by Example system because it embodies a mixed-initiative dialog – at any moment, either the user or the system may propose an action to be taken. The user and the system co-operate to define a "wrapper" – a description that describes how to extract example elements, like the price specification in an e-commerce Web page, from subsequent Web pages. This supplies the generalization for user actions.

6. COMPUTER-AIDED DESIGN OF CAD/CAM APPLICATIONS BY EXAMPLE

Perhaps the most direct analogy between Computer-Aided Design of products and interfaces is when those interfaces operated in the CAD/CAM domain itself. Patrick Girard [7] and his colleagues have developed some of the most sophisticated and industrial-strength Programming by Example systems as extensions to a traditional parametric CAD system for design of mechanical and electronic products. The systems, EBP and is successor GIPSE, notably incorporate constraints on the part descriptions. Constraints also play an important role in interface design. They paid careful attention to inference of control structures, and providing extensive example-oriented editing and debugging facilities.

Figure 4. EBP: A CAD/CAM system with PBE.

7. COMPUTER-AIDED EXTENSION OF USER INTERFACES BY EXAMPLE

Strictly speaking, many Programming by Example systems, like EBP mentioned above, do not create arbitrary interfaces from scratch in the way that the Interface Builder-like PBE systems do. Instead, they already come with a user interface (the underlying CAD/CAM system in EBP's case), and they add the ability to *extend* rather than completely redefine, the interface with new capabilities demonstrated by example. The original interface remains accessible to the user. So they, too, can be considered as Computer-Aided Design systems for User Interface Design, but by extension rather than by construction *de novo*.

Almost all PBE systems extend the interface in some way. Some just record a single macro and allow playback of the last macro recorded. Some allow the recorded sequences to be named or otherwise referred to. But the best in this category try to deal with the issue of how to integrate the user-defined capabilities seamlessly into the existing underlying application's interface metaphor.

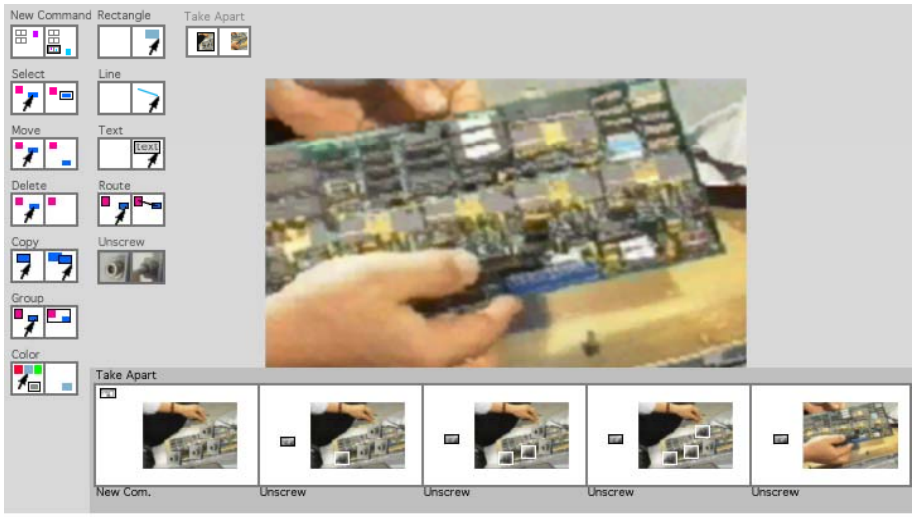

Figure 5. Mondrian.

My Mondrian system [10,11] represents the operations of the underlying graphical editor with "domino" icons that picture states of the screen before and after the operation. For example, the icon representing the "Draw Rectangle" operation consists of a left square showing a blank screen and a right square containing the rectangle. When the user defines a new operation, the system constructs a new icon composed of the screen state before the demonstration on the left side, and the screen state after the demonstration (when it is completed) on the right side. Thus the user-defined operation is represented with the same visual language as used for the built-in operations, and user-defined operations may then take place as parts of further user-defined operations.

Mondrian is also unique in its ability to define *declarative* as well as procedural knowledge by example. Mondrian lets the user introduce new generalization descriptions by drawing *graphical annotations* on an object. In an example domain of learning operational and maintenance procedures from video input, Mondrian allows the user to select portions of the video frame to serve as the graphical representations of example objects. Objects can be grouped, and labelled with text labels to show their structure. Mondrian then

learns part-whole hierarchies amongst the objects and the names of relations between the parts and wholes. When the user selects these objects in the future, Mondrian generalizes them according to their place in the part-whole hierarchy and their named relations. This is a way of introducing new *concepts* by example as well as simply new procedures.

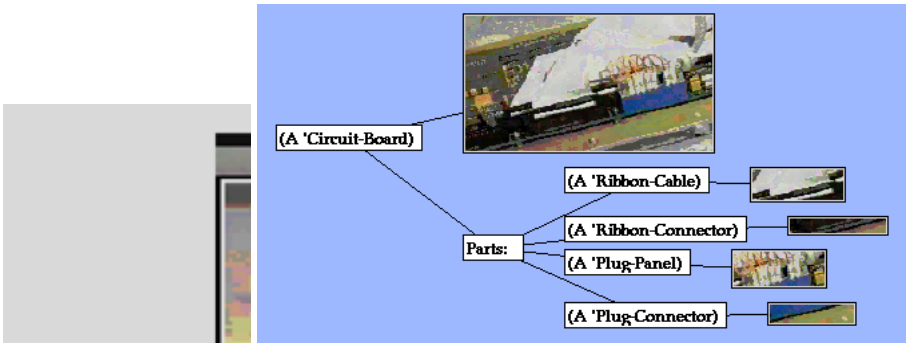

Figure 6. Graphical annotation in Mondrian, and inferred conceptual structure.

Another avenue for extension of user interfaces is through the use of *software agents* [2]. Interface agents provide proactive assistance to users of interactive interfaces by tracking user behavior, creating user profiles and user models, anticipating user needs, and accepting high-level goals and advice from the user. Agents effectively extend the capabilities of interfaces in dynamic ways not explicitly presented by the menu and icon operations, and in some cases, in ways not even anticipated by the interface designer.

Examples can play an important role in software agent interfaces because they can be the most effective means for a user to communicate their desires to the agent, obviating the need for more explicit instruction. We investigated an agent, Apple Data Detectors, that parses text occurring in naturally in e-mail, calendars and other applications, and applies semantically appropriate actions. We implemented a system, Grammex [13] for teaching the system new text patterns by presenting example text to be recognized and going through a dialog that interactively generalizes or specializes hypotheses about how to recognize substrings. This is the first interface that brings control of text recognition and parsing technology to non-expert users.

8. CONCLUSION

Like Computer-Aided Design systems for products and manufacturing applications, Computer-Aided Design systems for user interfaces can make the design process easier, more interactive and less-error prone. The success

of CAD/CAM systems for product design is predicated on the fact that the system allows the user to interact with virtual examples of the artifact under construction. This takes advantage of people's natural tendency to teach and to learn by example. But user interfaces have dynamic and interactive behavior, and to define such behavior we need more mechanism then just specification of appearance of interfaces. By recording actions of a simulated end-user and generalizing them, Programming by Example provides the technology for bringing the success of the CAD/CAM applications to user interface design.

REFERENCES

[1] Bauer, M., Dengler, D., and Paul, G., *Programming by Demonstration for Information Agents*, in [12], pp. 7-20.

[2] Bradshaw, J. (ed.), *Software Agents*, MIT Press, Cambridge, 1996.

[3] Buxton, W., Lamb, M.R., Sherman, D., and Smith, K.C., *Towards a Comprehensive User Interface Management System*, in Proc. of ACM Conf. on Computer Graphics SIGGRAPH'83 (Detroit, 25-29 July 1983), Computer Graphics, Vol. 17, No. 3, ACM Press, New York, 1983, pp. 576-583.

[4] Cypher, A. (ed.), *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, 1993.

[5] Finzer, W.F. and Gould, L., *Rehearsal World: Programming by Rehearsal*, in [4], pp. ?-?.

[6] Girard, P., *Bringing Programming by Demonstration to CAD Users*, in [12], pp. 135-162.

[7] Kahn, K., *Generalizing by Removing Detail: How any Program Can Be Created by Working with Examples*, in [12], pp. 21-44.

[8] Lieberman, H., *Constructing Graphical User Interfaces by Example*, in Proc. of Conf. Graphics Interface'82 (Toronto, May 1982), pp. 295-302.

[9] Lieberman, H., *Tinker: An Example Oriented Programming Environment for Beginning Programmers*, in R. Lawler and M. Yazdani (eds), Artificial Intelligence and Education, Vol. 1, Ablex Publishing Company, 1987.

[10] Lieberman, H., *Mondrian: A Teachable Graphical Editor*, in [4], pp. 340-358.

[11] Lieberman, H., *A Demonstrational Interface for Recording Technical Procedures by Annotation of Videotaped Examples*, International Journal of Human-Computer Studies, Vol. 43, 1995, pp. 383-417.

[12] Lieberman, H. (ed.), *Your Wish is My Command: Programming by Example*, Morgan Kaufmann, San Francisco, 2001. Introduction accessible at http://lieber.www.media.mit.edu/people/lieber/PBE/Your-Wish/

[13] Lieberman, H., Nardi, B., and Wright, D., *Training Agents to Recognize Text by Example*, in [12], pp. 227-244.

[14] Myers, B. and McDaniel, R., *Demonstrational Interfaces: Sometimes You Need a Little Intelligence; Sometimes You Need a Lot*, in [12], pp. 45-60.
[15] Myers, B., *Garnet: Uses of Demonstrational Techniques*, in [4], pp. 219-238.
[16] Myers, B., McDaniel, R., and Kosbie, D., *Marquise: Creating Complete User Interfaces by Demonstration*, in Proc. of ACM Conf. on Human Aspects in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 293-300.
[17] Repenning, A. and Perrone-Smith, C., *Programming by Analogous Examples*, in [12], pp. 351-370.
[18] Smith, D.C., Cypher, A., and Tesler, L., *Novice Programming Comes of Age*, in [12], pp. 7-19.
[19] Sugiura, A., *Web Browsing by Demonstration*, in [12], pp. 61-86.
[20] Wolber, D., *Pavlov: Where PBD Meets Macromedia's Director*, in [12], pp. 345-350.