September 2, 1999
RT0329
Human-Computer Interaction 11 pages

Research Report

A Survey of Demonstrational User Interfaces

Yoshinori Aoki

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Survey of Demonstrational User Interfaces

Yoshinori Aoki
IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato,
Kanagawa 242-8502, Japan
+81-462-73-4931
yoshia@jp.ibm.com

ABSTRACT
This paper summarizes research on demonstrational interfaces. With a demonstrational interface, the user performs actions, and the system then generates an abstract program. Such techniques are called *Programming by Demonstration* (PBD). PBD systems have been developed in many application domains. This paper classifies PBD systems on the basis of their application domains, and explains the features of various systems. One of the important techniques used in PBD systems is inductive inference. This paper describes the inference mechanism and interaction techniques used in PBD systems.

Keywords
Demonstrational User Interface, Programming by Demonstration, Programming by Example, Visual Programming, End-User Programming, Intelligent User Interface

1. INTRODUCTION

Nowadays, almost all computers have graphical user interfaces (GUIs). GUIs are user-friendly in that they are intuitive and free users from the need to input commands. Researchers in many fields, such as usability engineering, cognitive science, and industrial design, have made great efforts to improve user interfaces. Demonstrational interfaces have been developed to improve the usability of GUIs. This paper introduces systems that have demonstrational interfaces, called *Programming By Demonstration* (PBD) systems, and explains the interaction techniques used in them.

Demonstrational interfaces have been developed in many application domains. One of them is an interactive-application development environment. GUI programming is a complicated task for programmers, because they have to learn the event-driven model of the window system, and become familiar with the specifications of the library of GUI components. PBD systems allow users to create GUI programs by demonstrating the required operations, events, objects, and so on. This reduces the cost of GUI programming and allows programmers to concentrate on the application logic. PBD techniques have been developed by using the techniques of *Visual Programming* [Myers 86].

Some other PBD systems have been developed to automate users' operations. There are two ways of achieving this: (1) the user creates a macro program by demonstration, and (2) the system automatically detects the user's repetitive operations, and generates a macro. To realize the second approach, the system has to understand the user's intention from the operations. This approach allows end users to create a macro program without seeing any programs, because the system internally generates the macro program. End-user programming will be more important in the future [Myers 96], because no matter how successful interface designers are, systems will still need to be customized to the needs of particular users. It is important for end-user programming that the user not be required to see any program codes or scripts. To realize this objective, many interaction techniques have been developed for PBD systems.

The rest of the paper is organized as follows. The next section classifies PBD systems. In the section after that, I classify PBD systems on the basis of their application domains, and explain various PBD systems. Then, in the following section I summarize the techniques used in PBD systems. The last section presents conclusions.

2. A TAXONOMY OF PROGRAMMING BY DEMONSTRATION SYSTEMS

2.1. Terminology

Demonstrational interfaces allow a user to perform concrete actions while constructing an abstract program. Such techniques are called *Programming by Example* (PBE) or *Programming by Demonstration* (PBD) [Cypher 93]. PBE encompasses a number of approaches for creating programs by giving examples of their behavior or effects. Early systems attempted to create an entire program from examples of input-output pairs; this led to the term Programming by *Example*. Later systems allowed users to create programs by demonstrating the required operations, events, objects, and so on, and thus the term Programming by *Demonstration* was coined. However, both terms are used with the same meaning nowadays [Cypher 93]. In this paper, too, PBE and PBD are used interchangeably.

2.2. A Taxonomy

PBD systems can be classified into two groups according to the target users. The target users in one group are developers, while those in other group are end users. Early PBD systems were developed to help novice programmers. They were followed by interactive-application development systems, whose goal was to reduce programming costs. With these systems, users create GUI components by demonstration, or define the behaviors of GUI components by demonstration. PBD systems in the other group are designed to help end users. One category of systems in this second groups, sometimes called macro programming systems, can be used to create macros to automate their repetitive operations.

Myers classified PBD systems into intelligent interfaces and non-intelligent interfaces [Myers 92]. Intelligent interfaces have an inference mechanism, while non-intelligent interfaces do not. The inference mechanism is used to infer the user's intention and generate an abstract program. A PBD system that does not have an inference mechanism is sometimes called a *Programming With Example* system, and is distinguished from Programming By Example systems [Halbert 84] [Myers 86]. In this paper, I do not distinguish them, in line with many other recent researchers [Cypher 93].

3. PROGRAMMING BY DEMONSTRATION SYSTEMS

In this section, I classify PBD systems on the basis of their application domains, and explain various PBD systems.

3.1. Graphical Programming Environment

Early PBD systems were developed to help programmers. Although they were very simple, they inspired many HCI researchers who later developed PBD systems.

Pygmalion [Smith 75] [Smith 77] was the first programming by demonstration system. It was designed for programmers, and attempted to make programming easier. Pygmalion did not make inferences. Rather, it required the user to specify the program at an appropriate level of abstraction. One of its big advantages over traditional programming was that the user developed the program by operating on actual values in a particular instance of program execution.

Tinker [Lieberman 81] [Lieberman 93] is a system that permits a novice programmer to write Lisp programs by providing concrete examples of input data, and typing Lisp expressions or providing mouse input that gives the system directions on how to handle each example. The user explicitly indicates which objects should serve as examples, and may present multiple examples that illustrate conditional procedures. Tinker records the steps of the computation, and formulates a program for handling general cases. To perform an action, a user types in the name of a function and then selects its arguments from Tinker's Object List, which contains a list of objects. Tinker supports incremental program development: the user demonstrates a simple case and then later demonstrates more complicated situations. Programs are executed by typing Lisp expressions. The user constructs a program by using the generated programs. Functions defined by demonstration are invoked in exactly the same way as built-in Lisp functions. The Object List can be edited, or the resulting Lisp definition can be edited as text.

3.2. User Interface Development Systems

GUI programming is a complicated task, because a developer has to learn an event-driven model of the window system. The developer also has to become familiar with the properties and methods of each GUI object. PBD systems allow a novice programmer who is not familiar with GUI programming to create the GUI parts of an application by demonstration. In this way, the programmer can concentrate on the application logic and reduce the cost of GUI programming.

Peridot [Myers 90a] is an experimental tool for creating graphical, interactive user interface. Through *direct manipulation* [Shneiderman 83], users can create many types of user interface components. These are the low-level components of user interfaces (sometimes called "widgets"), and include most kinds of menus, property sheets, light buttons, radio buttons, scroll bars, and many other elements of GUIs. One of the primary goals of Peridot is to allow these interfaces to be created by non-programmers. To achieve this, it uses the techniques of Visual Programming [Myers 86], Programming by Example, Constraints [Leler 88], and inferencing. Peridot is one of the first systems to use Programming by Example with inferencing. Peridot handles two kinds of constraints: *graphical constraints* relate one graphic object to another, and *data constraints* ensure that a graphical object has a

particular relationship to a data value. Its inference mechanism is successful because it deals with a fairly narrow domain, the creation of user interface, and takes advantage of built-in knowledge about typical situations in that domain. The knowledge consists of simple 60 rules. The complete set of rules is provided in [Myers 88].

Garnet [Myers 90b] is a comprehensive user interface development environment in Lisp for X/11. It helps create graphical, interactive, direct manipulation interfaces. Garnet contains many high-level tools, including the Gilt interface builder [Myers 91c] [Hashimoto 92], the Lapidary interactive design tool [Myers 89], the C32 spreadsheet system [Myers 91a], the Jade dialog box system [Zanden 90], and Marquise [Myers 93]. Marquise is an ambitious tool, for it tries to allow users to create almost all parts of a user interface by demonstration, without any programming. Conventional interface builders have two modes: (1) edit mode, in which the user edits a presentation of the user interface, and (2) test mode, in which the user tests the behavior of the user interface. In addition, Marquise has two modes in which the user can define the behavior of the interface in response to the user's operations, such as mouse dragging, by demonstration. In [Myers 93], Myers shows that by using the lapidary, Gilt, and Marquise tools in Garnet, it is possible without any programming to create complete user interfaces of a graphic editor such as MacDraw or PowerPoint, as well as applications with various kinds of automatic layout for nodes.

Inference Bear [Frank 94a] is a user interface builder that has an inference engine specially designed for GUI interaction. The engine [Frank 94b] consists of two components: the *compactor* and the *inferencer*. When a user interacts with an interface object, the compactor specifies the target object, event type, and variables that are changed between the interaction, and then sends them to the inferencer. The inferencer has inference modules for each object type, and calls an appropriate inference module on the basis of the object type of the variable. Each inference module has its own knowledge and tactics for generating abstract programs. For example, a standard integer inference type comes with the engine, which derives a target variable (variables of *after* interaction) from a linear combination of source variables (variables of *before* interaction) and uses the standard Gaussian elimination to find a relation between the target variables and the source variables. The problem of Inference Bear is that it cannot deal with complicated interaction models. For example, when target variables (or source variables) consist of multiple object types, or the object type of the target variables and that of the source variables are different, the inference engine cannot work.

SILK [Landay 95] is a user interface development tool that supports the early stages of user interface prototyping. It allows designers to quickly sketch an interface by using an electronic pad and stylus. It then tries to recognize user interface widgets and other interface elements as they are drawn by the designer. As soon as a widget has been recognized, it can be used. For example, a sketched slider can be dragged up and down as soon as it has been recognized as a scrollbar widget. When the designer is happy with the interface, SILK will replace the sketches with real widgets and graphical objects; these can take on the specified look-and-feel of a standard graphical user interface, such as Motif, Windows, or Macintosh. SILK is unique in that it focuses on the early stages of interface prototyping. With SILK, interface designers can quickly develop prototypes by trial and error.

Gamut [McDaniel 97] is a PBD tool for building whole applications such as games. It uses many techniques for improving the interaction with the user [McDaniel 96]. For example, it defines some forms of interactions. One of them is *hints and nudges*: The user can actively nudge the system to give it hints that will help the system inference. Gamut focuses on games as its target application, and uses game-specific rules and inference algorithms [McDaniel 98]. In future, PBD techniques will be applied to more commercial applications, and domain-specific PBD techniques will then be needed to provide truly easy-to-use applications.

LEMMING [Olsen 95] is a tool that allows a user interface designer to create a new geometric widget by demonstration. Geometric widgets are those that are geometrically defined. A prime example is a scroll bar. Although a normal scrollbar does have event behavior, its primary model is the geometric manipulation of the thumb. The position of the thumb is tied to the value being controlled by the widget. LEMMING provides general-purpose mechanisms for mapping between the presentation of a geometric widget and its control value. First, a user draws a set of widgets and gives a set of example data to LEMMING. Then, it automatically learns the mapping between the presentation and the control value. I think the most important feature of LEMMING is that it allows a designer to create complicated widgets, not only 2D but also 3D. Almost all PBD systems support only traditional widgets such as buttons, radio buttons, and menus; consequently, it is difficult to create complicated interface components. It will be more important for PBD systems to support not only traditional graphical user interfaces but also next-generation user interfaces such as 3D interfaces.

In Druid UIMS [Singh 90], a user can indicate the location of a widget or dialog by demonstration. [Slagle 94] presents the idea for creating GUI program automatically from operation histories on Xf interface builder, which runs on the X window system, and Tcl/Tk toolkit.

3.3. Interactive-application development Systems

Wolbert et al. developed user interface development systems based on a role-playing methodology called *stimulus-response demonstration*. DEMO [Wolbert 91] is a user interface development system that uses PBD techniques and is based on the stimulus-response model. In DEMO, a developer draws an interface by using a drawing editor, and specifies the interactive behavior of the interface by directly demonstrating how the system should respond to stimuli from an end-user. To specify interactive behavior, the developer first plays the role of the end-user and performs some graphical actions, then plays the role of the system and performs the graphical actions that should be executed in response to the end-user's stimulus. The system generalizes from stimulus-response demonstrations to generate stimulus-response specifications that define the behavior of the interface. DEMO uses interactive dialog with the developer to direct its inferencing choices.

Fisher et al. developed DEMO II [Fisher 92] to increase DEMO's ability to draw inferences about interface behavior without having to rely on dialog with the developer. To accomplish this, they integrated a rule-based reasoning component into DEMO. The reasoning component is implemented by means of expert system techniques. It has approximately 75 rules on object naming, constraint identification, constraint refinement, constraint satisfaction, and communications and control. With the component, DEMO II can gradually modify a program from multiple examples.

DEMO and DEMO II were successful systems for creating static user interfaces. Wolbert integrated a time management function into the stimulus-response model, to allow the creation of animated user interfaces. He developed Pavlov [Wolbert 96], which can handle time as one type of stimulus. With Pavlov, users can create animated interfaces without programming. An important aspect of a PBD system is how a user edits the behaviors inferred by the system. Pavlov's editor provides a view of multiple time-lines: one for the events that occur without an end-user stimulus, and one for the events triggered by each end-user stimulus that has been demonstrated. The problem with Pavlov is that it does not have a mechanism for linking created user interfaces and application programs.

KidSim [Smith 94] [Cypher 95] is an environment that allows children to create their own simulations. They create their own characters, and create rules that specify how the characters are to behave and interact. Each character has a list of rules, and acts on a *game board* divided into discrete spaces, like a checkerboard. When a user clicks on a character's "New Rule" button, the entire board darkens, except for a spotlight around the character. The user reshapes this spotlight to specify the context for the new rule, and demonstrates what the rule should do by moving the objects in the spotlight. Apple Computer produced a commercial product call *Cocoa* from KidSim.

3.4. Automating and Prediction of a User's Operations

MIKE [Olsen 88] is the first User Interface Management System (UIMS) that supports macro programming functions. A user can define a macro as a set of existing commands and register it in a menu. A user can also define conditional branches by using a macro editor. The GUI model of MIKE is very primitive compared with current major GUI models, which support interaction with a user in an event-driven manner. MIKE merely enables a user to call a set of functions from a menu.

SmallStar [Halbert 84] [Halbert 93] is a macro programming system using PBD techniques, and is based on the Xerox Star system. A user creates a program by clicking the *Start Recording* and *Stop Recording* buttons. SmallStar introduced the idea of *data descriptions*, which describe the data to be used when the program is recorded. SmallStar uses no inference, and chooses an initial data description for every object used during recording. The recorded program is displayed in a scripting language that includes icons for various types of objects, such as folders and text selections. After recording, the user can identify the target objects by editing a *data description sheet*. SmallStar also introduced the idea of a *program icon*, which represents the recorded program.

LEDA [Mima 91] is a programming environment for developing a PBD application. An application developed with LEDA provides functions for recording and playback of a user's operations. The programmer does not have to develop recording and playback functions for each application. Such applications are sometimes called macro programming systems. In conventional macro programming systems, there was no general method for specifying objects when the target object is different from the object specified at the recording time. LEDA solved the problem by enhancing the data descriptions of SmallStar [Halbert 84] [Halbert 93] and enabling the user to explain how to choose a target object by using a data description mechanism. LEDA is implemented on top of the window manager of IBM OS/2. AIDE [Piernot 93] is also an application-independent PBD system that allows a developer to add advanced macro capabilities to a Smalltalk-based application without re-implementing everything from scratch.

Almost all PBD systems use application data to specify objects or detect a user's operations. Triggers [Potter 93] is unique in that it uses only bitmap data displayed on a screen, and is therefore applicable to almost all applications that display graphics on a screen.

Eager [Cypher 91] is a PBD system that automatically detects a user's repetitive operations on HyperCard, which runs on Macintosh PCs. When Eager detects repetitive operations by a user, it displays an Eager icon and highlights the GUI object that is expected to be selected in the next operation. If the user interacts with the highlighted GUI object, Eager considers that the detection is correct. It then highlights the next GUI object that is expected to be selected. If the user confirms that Eager's predictions are correct, he or she can automatically execute the repetitive operations by clicking an Eager icon. One of the most important features of Eager is that users do not have to perform any operations in addition to normal operations. They do not have to specify a starting or an ending point for repetitive operations, and they do not see any programs. All they have to do is click an Eager icon to confirm that Eager's predictions are correct. If Eager's prediction is incorrect, they can ignore the Eager icon and continue operations.

Pursuit [Modugno 94] allows a user to create shell programming by demonstration. The user can easily understand the program, because Pursuit displays before-and-after pair of icons that represent the status of target objects.

DemoOffice [Sugiura 96] is a PBD system that integrates an e-mail system and a relational database. It extracts necessary operations from the recorded history, automatically generates a macro to make it applicable to future contexts, and selects arguments for the macro. A generated macro is invoked by clicking a macro button located on a macro bar. Since almost all the processes in macro definition are performed automatically, it is important that users be able to confirm what the macro does. When the user moves a mouse pointer over the macro button, DemoOffice explains with a tip-help how the macro behaves, and displays an example of its execution.

3.5. Text Editing

Editing by Example [Nix 85] is a system that infers text-transformation rules from before-and-after pairs of text given by a user, and applies those rules to the rest of the text. Editing by Example handles only pairs of text, while almost all PBD systems treat users' operations as example data.

TELS [Mo 92] generates a generalized program from a user's repetitive operations. It handles only four kinds of operations; paste, delete, move, and select. For example, If the user selects the telephone numbers 222-3456 and 234-5555, TELS generates a program that selects 2??-??5?, where '?' represents any one-digit integer. TELS supports incremental program modification: if a user corrects the proposed text, TELS modifies the program to improve the heuristic algorithm it uses for inference.

Dynamic Macro [Masui 94a] extracts a repetitive operation pattern and generates a macro when a user instructs the system. The user does not have to manually register the macro, although the user has to tell the system to generate a macro. It runs on GNU Emacs and can handle all Emacs operations.

POBox [Masui 98a] [Masui 98b] is a pen-based Japanese text input method that integrates software keyboards, handwriting recognition, and PBD. If a user taps on the "ma" key and immediately releases the pen, the system shows candidate words that begin with the pronunciation "ma". If the user taps on the "ma" key and waits a while, a pull-down menu appears and shows candidate words around the pen position. When the user touches the tablet with the pen, the system starts handwriting recognition and interprets the strokes incrementally, and shows candidate words that begin with the strokes. Masui calls the approach *Composition By Example*.

Tourmaline [Myers 91b] provides a macro generation function from a user's demonstration. The system generates a style by using heuristics when a user gives a format example such as a font, size, position, and so on.

3.6. Graphics Editing

Metamouse [Maulsby 89a] [Maulsby 89b] is a system that generates a program from a user's graphical editing. When Metamouse detects a repetition, it offers to perform actions automatically. The user can verify each action, undo it, and correct it. Metamouse uses a metaphorical mouse, that is, a graphical turtle named Basil, to show the actions. The user explains his/her intent to Basil through a *teaching metaphor*. Metamouse is similar to Eager [Cypher 91] in that both systems predict looping patterns over history. In addition to looping patterns, Metamouse can also find branching patterns when a user performs a different action from the one expected.

Chimera [Kurlander 92] is a macro programming system for a graphic editor. In SmallStar, a user can edit a generated program as text. Chimera shows a generated macro as a sequence of operation icons, and a user can visually edit them. Each operation icon graphically represents an operation. Mondrian [Lieberman 92] is also a macro programming system for a graphic editor that shows a generated macro as an operation icon. Mondrian's operation icon presents a before-and-after pair of the images for the operation. A user can visually reuse generated macros through the icons. Mondrian teaches the user about its inferencing processes through voice output or in natural-language text when it generates a macro.

PBD techniques are also used in graphics-layout applications. Layout By Example [Hudson 93] is a system that infers graphics-layout rules from example layouts. The system allows a user to choose one example from

multiple examples that are the results of the inferencing. The user can let the system infer again to correct its inferencing. In this approach, the system corrects its inferencing step by step. The approach allows the system to improve the inferencing effectively through a little interaction with the user, while conventional PBD systems allow users to give multiple examples at one time.

TRIP2 [Matsuoka 92] and TRIP3 [Miyashita 92] are declarative graphics-layout applications using graphical constraints. TRIP2 realizes bidirectional translation between a set of application data and its visual representation by using a set of declarative translation (mapping) rules. Through bidirectional translation, TRIP2 can not only automatically modify the mapping from the visual representation to its application data, but also the one from the application data to its visual representation, when the user moves a graphical object. TRIP3 can generate declarative translation rules from multiple example visualizations. In TRIP3, a user gives multiple examples at one time, and the system infers general translation rules by comparing the examples. It then shows the user its inferencing, and the user corrects any mistakes. However, it is difficult to reduce mistakes in inferencing from a small number of examples. IMAGE [Miyashita 94] was developed to solve this problem. In IMAGE, a user gives one example at a time, and the system shows the user its inferencing by displaying another example to which the generated translation rules have been applied. The user then interactively modifies the example to correct the inferencing.

Masui developed a graph-layout system that learns a user's preferences from examples [Masui 94b]. In the system, the user shows the system pairs of good and bad layout examples, and the system infers an evaluation function using the *genetic programming* technique [Kozierok 93]. The system is unique in that it infers a user's preferences, which infers user preferences that are difficult to express in text or a program.

When users want to create a chart or graphic from data in a table, they can do so in with Lotus 1-2-3 or Microsoft Excel by selecting from a menu of pre-defined chart types. However, they cannot define a new type of a chart. Gold [Myers 94] allows a user to create a complex business chart by providing examples. The user draws an example of a graphic object and demonstrates to the system correspondences between data in a table and the properties of the graphic object. The system then create a chat for a set of actual data.

Sage [Roth 90] is another system that creates charts by analyzing examples. It consists of a visualization tool called SageBrush [Roth 94] and a knowledge base called SageBook [Roth 94]. Rather than having users draw examples of the desired display, in SageBrush the user assembles displays by selecting graphical objects and assembles data to match their properties.

3.7. World Wide Web

Internet Scrapbook [Sugiura 98a] [Sugiura 98b] is an application that provides a function for automating repetitive browsing tasks by means of PBD techniques. An interesting function is that the system allows a user to create a personal page by clipping only the necessary portions from multiple Web pages. The personal page is automatically updated by the system.

Turquoise [Miller 97] is another application that allows a user to automate repetitive browsing tasks by demonstration. The user can create a personal page by clipping only the necessary portions from multiple Web pages, in the same way as Internet Scrapbook [Sugiura 98a] [Sugiura 98b]. Turquoise has a *pattern matcher* designed to find portions of an HTML document. It infers patterns automatically from the user's demonstration by using a heuristic knowledge base of pattern templates, which are patterns containing placeholders, such as "<HTML element> **after** Heading **containing** <text>." Turquoise allows users to add patterns by themselves.

Web Operation Recorder [Aoki 98] allows a user to record operations on a Web browser, and to play the recorded operations with a real Web browser. It runs in a Java-enabled Web browser. Users do not have to prepare the Web contents to be recorded; they can work with ordinary Web pages. In addition, Web Operation Recorder allows a user to add explanations to existing HTML contents by making "ink" annotations and attaching text, images, and hyperlinks to a Web page [Aoki 99].

4. INTERACTION TECHNIQUES USED IN PBD SYSTEMS

Many interaction techniques have been developed for PBD systems, as explained in the previous section. This section describes the key techniques used in successful PBD systems.

4.1. Technical Issues in PBD Systems

The following is a typical flow of a PBD system:

(1) A user first demonstrates, to provide the system with examples.

(2) The system then generates programs. Many recent PBD systems use an inference mechanism to generalize a generated program.

(3) Some PBD systems show the generated program to the user. The user then modifies the program to correct the mistakes in the inferencing.

Inference mechanisms are used in many PBD systems to generates an abstract program in step (2). Inferencing is one of the key techniques in PBD systems, because the

flexibility of the generated program depends on the inference mechanism.

In step (3), the visualization technique is important. One of the problems with early PBD systems was that users had to check the generated program in text, and thus had to know the programming language. In addition, understanding programs written by someone else is a time-consuming task. Therefore, the total cost of programming cannot be reduced. Another important issue is how a user can modify generated programs without editing them.

The following subsection describes some of the PBD system interaction techniques developed to resolve the above issues.

4.2. Inference Mechanism

Peridot [Myers 90a] was one of the first PBD systems to use an inference mechanism. Subsequently, many such systems were developed. Inferencing is used to understand a user's intent from his/her operations and generate an abstract program. Peridot infers graphical constraints and data constraints by using built-in knowledge of a specified GUI domain. Eager [Cypher 91] is another successful system that has an inference mechanism using domain knowledge.

Inference Bear [Frank 94a] has an inferencing engine [Frank 94b] that contains no domain knowledge. When a user interacts with an interface object, the engine draws inferences by comparing before-and-after snapshots of the object. The advantage of having no domain knowledge is that the engine is flexible and can be used for many domains. It can also be used at any level of abstraction, and is extensible. The engine comes with three predefined inference types: string, integer, and boolean variables. A user can easily add other inference types. The disadvantage of no domain knowledge is that the engine cannot make use of having such knowledge in the inference process. Hence, a user has to give many demonstrations, or manually modify the programs.

The main feature of the inference mechanism of Metamouse [Maulsby 89a] [Maulsby 89b] is that it generates conditional branches from multiple examples. DEMO II's inference mechanism can also generate a program that has conditional branches [Fisher 92]. In DEMO II, a user gives multiple examples to the system. The inference mechanism then extracts graphical constraints from the examples, and generates conditional branches from the differences between the constraints. When a system can generate a program that has conditional branches, the program can be more flexibile.

4.3. Visualization and Modification of Generated Programs

Peridot [Myers 90a] shows each inference in a text message, using a natural-language template, and asks the user to verify it. Thus, the user can find a mistake in the inference and correct it. However, it is very time-consuming for the user to answer each inference message.

Metamouse [Maulsby 89a] [Maulsby 89b] shows a graphical turtle named Basil to show detected operations to the user. When the system detects a user's repetitive operations, it offers to perform them automatically. If the user accepts the offer, Basil performs as a metaphorical mouse and the user can verify the detected repetitive operations by watching Basil's performance. If the user is not satisfied with Basil's performance, he/she can undo the operations. This seems to be a natural approach, because the system also shows detected operations by Basil's demonstration.

If Eager [Cyper 91] detects a user's repetitive operations, the system shows an Eager icon and highlights the object that is expected to be selected in the user's next operation. The main feature of Eager's approach is that Eager does not use any text message or dialog box to show the system's inference. Highlighting the object is intuitive, and does not interrupt the user's operations even if the prediction is wrong. If the user selects the highlighted object, Eager deduces that the prediction is correct. The fact that Eager does not bother users is important. Microsoft Office also uses the same approach, providing some input assistant functions. For example, when a user types the same text in several cells on Excel, Excel shows the colored text in the cell. If the user stops typing and presses the Enter key, Excel completes the input of the text. If the colored text does not satisfy the user, the user can ignore the colored text and continue typing.

Like Eager, Edward [Bos 92] also detects a user's next operation. Its inference mechanism is similar to that of Eager. However, it is unique in that the user can specify operations in natural language. It also shows the results of inferencing and asks questions of the user in natural language.

Chimera [Kurlander 92], Mondrian [Lieberman 92], and Pursuit [Modugno 94] generate macro programs and show them as operation icons. Chimera has a component called Macro Builder, in which snapshots of the user's operations are shown and can be visually edited. In Mondrian and Pursuit, an operation icon includes a before-and-after pair of operation snapshots. The user can visually reuse generated macros through the icons. Mondrian is unique in that it tells the user the result of its inferencing by using a speech engine.

Masui's graph-layout system [Masui 94b] infers the user's preferences from example layouts. The system does not directly show a generated evaluation function to the user; instead, it shows another example layout to which the

evaluation function has been applied. The user can modify the evaluation function by judging the example layout.

5. CONCLUSIONS

Programming By Demonstration systems have been developed for many application domains. Early PBD systems were intended to help application developers, and therefore assumed that users could understand and modify the generated programs. These systems were successful, and have influenced current commercial visual programming environments such as Microsoft Visual Basic. Since then, many PBD systems have been developed to help the end users. The following issues should be considered in future research on PBD systems:

- PBD systems should make good use of domain knowledge to reduce the interaction with users.
- PBD techniques should be merged into non-PC devices and applied to a new user interface paradigm.

I believe that it will be increasingly important to apply PBD techniques to commercial products to help end users. Microsoft eagerly adopts PBD techniques in its commercial products. For example, Excel detects a user's repetitive text input and complements the input. Microsoft Office also has a macro recording functions. These PBD techniques are very helpful to users, although they are simple. To help end users, PBD components have to be naturally merged into applications. Conventional PBD systems require users to interact with them in order to correct their inferences. This approach is effective if the target users are computer experts; however, such interaction bothers or embarrasses many end users. The system should collect more information from ordinary operations. However, it is impossible for a system to perfectly infer a user's intention from ordinary operations, and therefore domain knowledge will become more important. Acquisition of domain knowledge from operations will also be important, to complement built-in domain knowledge.

Another issue is that all existing PBD systems have been developed for graphical-user-interface environments, sometimes called *WIMP* (Window, Icon, Menu, and Pointing device) environments. PBD techniques should be applied to other interface environments. For example, many new kinds of devices, such as information appliances, PDAs, and intelligent cellular phones, are being developed. Such devices require simpler user interfaces than the GUI of PCs, because (1) they have poorer displays than PCs, (2) they have only simple input devices, and (3) the target users are end users. PBD techniques can reduce users' complicated inputs on such devices. However, new interaction techniques will be needed, because such devices may not use WIMP environments.

REFERENCES

[Aoki 98] Aoki, Y., Ando, F., Nakajima, A., "Web Operation Recorder and Player," IBM TRL Research Report RT0267, 1998.

[Aoki 99] Aoki, Y., Nakajima, A., "User-Side Web Page Customization," *Proceedings of 8th International Conference on Human Computer Interaction (HCI International '99)*, to be published in August 1999.

[Bos 92] Bos, E., "Some Virtues and Limitations of Action Inferring Interfaces," *Proceedings of UIST '92*, ACM Press, NY, pp. 79-88, November 1992.

[Cypher 91] Cypher, A., "Eager: Programming Repetitive Tasks by Example," *Proceedings of CHI '91*, ACM Press, NY, pp. 33-39, April 1991.

[Cypher 93] Cypher, A., Ed., *Watch What I Do: Programming by Demonstration*, The MIT Press, Cambridge Mass., 1993.

[Cypher 95] Cypher, A., "KidSim: End User Programming of Simulations," *Proceedings of CHI '95*, ACM Press, NY, pp. 27-34, May 1995.

[Fisher 92] Fisher, G., Busse, D. E., "Adding Rule-Based Reasoning to a Demonstrational Interface Builder," *Proceedings of UIST '92*, ACM Press, NY, pp. 89-97, November 1992.

[Frank 94a] Frank, M., Foley, J., "Inference Bear: Inferring Behavior from Before and After Snapshots," Technical Report git-gvu-94-12, Georgia Institute of Technology, Graphics, Visualization and Usability Center, April 1994. Available at http://www.gatech.edu/gvu/gvutop.html.

[Frank 94b] Frank, M. R., Foley, J. D., "A Pure Reasoning Engine for Programming by Demonstration," *Proceedings of UIST '94*, ACM Press, NY, pp. 95-101, November 1994.

[Halbert 84] Halbert, D., "Programming by Example," Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of California Berkeley, 1984. Also as: Halbert, D., "Programming by Example," Technical Report No. OSD-T8402, Office Systems Division, Xerox Corporation, 1984.

[Halbert 93] Halbert, D. C., "SmallStart: Programming by Demonstration in the Desktop Metaphor," Chapter 5, pp. 103-123. In Cypher [Cypher 93], 1993.

[Hashimoto 92] Hashimoto, O., Myers, B. A., "Graphical Styles for Building User Interaces by Demonstration," *Proceedings of UIST '92*, ACM Press, NY, pp. 117-124, November 1992.

[Hudson 93] Hudson, S. E., His, C. N., "A Synergistic Approach to Specifying Simple Number Independent Layout by Example," *Proceedings of CHI '93*, ACM Press, NY, pp. 285-292, April 1993.

[Kurlander 92] Kurlander, D., Feiner, S., "A History-Based Macro by Example System," *Proceedings of UIST '92*, ACM Press, NY, pp. 99-106, November 1992.

[Kozierok 93] Kozierok, R., Maes, P., "A Learning Interface Agent for Scheduling Meetings," *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*, ACM Press, N.Y., pp. 81-88, January 1993.

[Landay 95] Landay, J. A., Myers, B. A., "Interactive Sketching for the Early Stages of User Interface Design," *Proceedings of CHI '95*, ACM Press, NY, pp. 43-50, May 1995.

[Leler 88] Leler W., *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, Reading, MA, 1988.

[Lieberman 81] Lieberman, H., "Tinker: Example-Based Programming for Artificial Intelligence," *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1060, August 1981.

[Lieberman 92] Lieberman, H., "Dominos and Storyboards: Beyond Icons on Strings," *Proceedings of the 1992 Workshop on Visual Languages*, IEEE, pp. 65-71, September 1992.

[Lieberman 93] Lieberman, H., "Tinker: A Programming by Demonstration System for Beginning Programmers," Chapter 2, pp. 49-64, In [Cypher 93], 1993

[Masui 94a] Masui, T., Nakayama, K., "Repeat and Predict – Two Keys to Efficient Text Editing," *Proceedings of CHI '94*, ACM Press, NY, pp. 118-123, April 1994.

[Masui 94b] Masui, T., "Evolutionary Learning of Graph Layout Constraints from Examples," *Proceedings of UIST '94*, ACM Press, NY, pp. 103-108, November 1994.

[Masui 98a] Masui, T., "An Efficient Text Input Method for Pen-Based Computers," *Proceedings of CHI '98*, ACM Press, NY, pp. 328-335, April 1998.

[Masui 98b] Masui, T., "Integrating Pen Operations for Composition by Example," *Proceedings UIST' 98*, ACM Press, NY, pp. 211-212, November 1998.

[Matsuoka 92] Matsuoka, S., Takahashi, S., Kamada, T., Yonezawa, A., "A General Framework for Bidirectional Translation between Abstract and Pictorical Data," *ACM Transactions on Information Systems*, Vol. 10, No. 4, pp. 408-437, October 1992.

[Maulsby 89a] Maulsby, D. L., Witten, I. H., and Kittlitz, K. A., "Metamouse: Specifying Graphical Procedures by Example," *Proceedings of SIGGRAPH '89*, ACM Press, NY, pp. 127-136, July-August 1989.

[Maulsby 89b] Maulsby, D. L., Witten, I. H., and Kittlitz, K. A., "Inducing Programs in a Direct-Manipulation Environment," *Proceedings of CHI '89*, ACM Press, NY, pp. 57-62, April-May 1989.

[McDaniel 96] McDaniel, R. G., "Improving Communication in Programming-by-Demonstration," *Proceedings of CHI '96 Conference Companion*, ACM Press, NY, pp. 55-56, April 1996. Available at http://www.cs.cmu.edu/afs/cs/user/richm/public/www/-doct96.html.

[McDaniel 97] McDaniel, R. G., Myers, B. A., "Gamut: Demonstrating Whole Applications," *Proceedings of UIST '97*, ACM Press, NY, pp. 81-82, October 1997.

[McDaniel 98] McDaniel R. G., Myers, B. A., "Building Applications Using Only Demonstration," *Proceedings of Intelligent User Interface (IUI) '98*, ACM Press, NY, pp. 109-116, January 1998.

[Miller 97] Miller, R. C., Myers, B. A., "Creating Dynamic World Wide Web Pages By Demonstration," Technical Report #CMU-CS-97-131, School of Computer Science Carnegie Mellon University, 1997. Available at ftp://reports.adm.cs.cmu.edu/usr/anon/-1997/CMU-CS-97-131.ps

[Mima 91] Mima, Y., "A Visual Programming Environment for Programming by Example Abstraction," *Proceedings of the 1991 Workshop on Visual Languages*, IEEE, pp. 132-137, October 1991.

[Miyashita 92] Miyashita, K., Matsuoka, S., Takahashi, S., "Declarative Programming of Graphical Interfaces by Visual Examples," *Proceedings of UIST '92*, ACM Press, NY, pp. 107-116, November 1992.

[Miyashita 94] Miyashita, K., Matsuoka, S., Takahashi, S., Yonezawa, A., "Interactive Generation of Graphical User Interfaces by Multiple Visual Examples," *Proceedings of UIST '94*, ACM Press, NY, pp. 85-94, November 1994.

[Mo 92] Mo, D. H., Witten, I. H., "Learning Text Editing Tasks from Example: A Procedural Approach," *Behavior & Information Technology*, Vol. 11, No. 1, pp. 32-45, 1992.

[Modugno 94] Modugno, F., Myers, B. A., "A State-Based Visual Language for a Demonstrational Visual Shell," *Proceedings of the 1994 IEEE Symposium on Visual Languages*, IEEE, pp. 304-311, October 1994.

[Myers 86] Myers, B. A., "Visual Programming, Programming by Example, and Program Visualization: A Taxonomy," *Proceedings of CHI '86*, ACM Press, NY, pp. 59-66, April 1986.

[Myers 88] Myers, B. A., *Creating User Interface by Demonstration*, Academic Press, San Diego, 1988.

[Myers 89] Myers, B. A., Zanden, B. V., Dannenberg, R., "Creating Graphical Interactive Application Objects by Demonstration," *Proceedings of UIST '89*, ACM Press, NY, pp. 95-104, November 1989.

[Myers 90a] Myers, B. A., "Creating User Interfaces Using Programming by Example, Visual Programming and Constraints," *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 2, pp. 143-177, April 1990.

[Myers 90b] Myers, B. A., Giuse, D., Dannenberg, R., Zanden, B. V., Kosbie, D., Pervin, E., Mickish, A., Marchal, P., "Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces," *IEEE Computer*, Vol. 23, No. 11, pp. 71-85, November 1999.

[Myers 91a] Myers, B. A., "Graphical Techniques in a Spreadsheet for Specifying User Interfaces," *Proceedings of CHI '91*, ACM Press, NY, pp. 243-249, April 1991.

[Myers 91b] Myers, B. A., "Text Formatting By Demonstration," *Proceedings of CHI '91*, ACM Press, NY, pp. 251-256, April 1991.

[Myers 91c] Myers, B. A., "Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs," *Proceedings of UIST '91*, ACM Press, NY, pp. 211-220, November 1991.

[Myers 92] Myers, B. A., "Demonstrational Interfaces: A Step beyond Direct Manipulation," *IEEE Computer*, Vol. 25, No. 8, pp. 61-73, August 1992.

[Myers 93] Myers, B. A., McDaniel, R. G., Kosbie, D. S., "Marquise: Creating Complete User Interfaces by Demonstration," *Proceedings of CHI '93*, ACM Press, NY, pp. 293-300, April 1993.

[Myers 94] Myers, B. A., Goldstein, J., Goldberg, M. A., "Creating Charts by Demonstration," *Proceedings of CHI '94*, ACM Press, NY, pp. 106-111, April 1994.

[Myers 96] Myers, B. A., Hollan, J., Cruz, I., "Strategic Directions in Human Computer Interaction," *ACM Computing Surveys*, Vol. 28, No. 4, pp. 794-809, December 1996.

[Nix 85] Nix, R. R., "Editing by Example," *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 4, pp. 600-621, October 1985.

[Olsen 88] Olsen, D. R. Jr., Dance, J. R., "Macros by Example in a Graphical UIMS," *Computer Graphics & Applications*, IEEE, Vol. 8, No. 1, pp. 68-78, January 1988.

[Olsen 95] Olsen, D. R., Jr., Ahlstrom, B., Kohlert, D., "Building Geometry-Based Widgets by Example," *Proceedings of CHI '95*, ACM Press, NY, pp. 35-42, May 1995.

[Piernot 93] Piernot, P. P., Yvon, M. P., "The AIDE Project: An Application-Independent Demonstrational Environment," Chapter 18, pp. 383-401, In [Cypher 93], 1993.

[Potter 93] Potter, R., "Triggers: Guiding Automation with Pixels to Achieve Data Access," Chapter 17, pp. 361-380, In [Cypher 93], 1993.

[Roth 90] Roth, S. F., Matthis, J., "Data Characterization for Intelligent Graphics Presentation," *Proceedings of CHI '90*, ACM Press, NY, pp. 193-200, April 1990.

[Roth 94] Roth, S. F., Kolojejchick, J., Matthis, J., Goldstein, J., "Interactive Graphic Design Using Automatic Presentation Knowledge," *Proceedings of CHI '94*, ACM Press, NY, pp. 112-117, April 1994.

[Shneiderman 83] Shneiderman B., "Direct Manipulation: A Step beyond Programming Language," *IEEE Computer*, Vol. 16, No. 8, pp. 57-69, August 1983.

[Singh 90] Singh, Kok, C. H., Ngan, T. Y., "Druid: A System for Demonstrational Rapid User Interface Development," *Proceedings of UIST '90*, ACM Press, NY, pp. 167-177, October 1990.

[Slagle 94] Slagle, J. R., Wieckowski, Z., "Ideas for Intelligent User Interface Design," *Tcl '94 Workshop Proceedings*, 1994.

[Smith 75] Smith, D. C., "Pygmalion: A Creative Programming Environment," Report No. STAN-CS-75-499, Department of Computer Science, Stanford University, 1975.

[Smith 77] Smith, D. C., *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*, Birkhauser, Basel, 1977.

[Smith 94] Smith, D. C., Cypher, A., Spohrer, J. C., "KIDSIM: Programming Agents without a Programming Language," *CACM*, Vol. 37, No. 7, ACM Press, NY, pp. 54-67, July 1994.

[Sugiura 96] Sugiura, A., Koseki, Y., "Simplifying Macro Definition in Programming by Demonstration," *Proceedings of UIST '96*, ACM Press, NY, pp. 173-182, November 1996.

[Sugiura 98a] Sugiura, A., Koseki, Y., "Internet Scrapbook: Automating Web Browsing Tasks by Programming-by-Demonstration," Proceedings of *7th International World Wide Web Conference*, April 1998. Available at http://www7.scu.edu.au/programme/posters/1886/-com1886.htm

[Sugiura 98b] Sugiura, A., Koseki, Y., "Internet Scrapbook: Automating Web Browsing Tasks by

Demonstration," *Proceedings of UIST '98*, ACM Press, NY, pp. 9-18, November 1998.

[Wolber 91] Wolber, D., Fisher, G., "A Demonstrational Technique for Developing Interfaces with Dynamically Created Objects," *Proceedings of UIST '91*, ACM Press, NY, pp. 221-230, November 1991.

[Wolber 96] Wolber, D., "Pavlov: Programming by Stimulus-Response Demonstration," *Proceedings of CHI '96*, ACM Press, NY, pp. 252-259, April 1996.

[Zanden 90] Zanden, B. V., Myers, B. A., "Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces," *Proceedings of CHI '90*, ACM Press, NY, pp. 27-34, April 1990.