RC23116 (W0402-098) February 20, 2004
Computer Science

IBM Research Report

Programming-by-Demonstration for Behavior-based User Interface Customization

Lawrence D. Bergman, Tessa A. Lau, Vittorio Castelli, Daniel Oblinger
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Programming-by-Demonstration for Behavior-based User Interface Customization

Lawrence D. Bergman, Tessa A. Lau, Vittorio Castelli, Daniel Oblinger
IBM T.J. Watson Research Center
30 Sawmill River Rd.
Hawthorne, NY 10549 USA
+1 914 784 7946
bergman@watson.ibm.com

ABSTRACT

Programming by demonstration (PBD) is a powerful tool for creating new user interface controls by capturing user behavior. A programming by demonstration system watches what the user does, infers the user's intent, and on request, performs actions on the user's behalf. Personal Wizards is a desktop programming by demonstration system running on Windows platforms. We describe the Personal Wizards PBD system, and speculate on several ways in which existing user interfaces could be customized with new controls.

Keywords

Programming by demonstration, user interface customization, adaptive interfaces, machine learning

INTRODUCTION

In this paper we explore the use of programming by demonstration (PBD) as a technique for behavior-based user interface customization, and present Personal Wizards, a system that implements programming by demonstration.

Programming by demonstration [Cypher93, Lieberman01] creates a procedure by capturing a sequence of user actions (e.g., key presses and mouse clicks) on an interface. A procedure can be considered to be a sequence of context-dependent actions that accomplish a particular well-defined task. Examples include: filling out a travel expense form, troubleshooting a network card, and reserving a conference room using an online reservation system.

Note that PBD only permits the creation of procedures that invoke a sequence of already-defined interface controls. One can think of PBD as creating new controls from old – as new procedures are assembled and bound to new controls, these new controls, in turn, become available as components of future procedures. Creating a novel action that cannot be composed from existing UI actions, however, requires more traditional software development techniques and tools.

The rest of this paper will discuss programming by demonstration in more detail, and describe Personal Wizards, a desktop PBD system. We will conclude with a discussion of how PBD-based actions could be used to customize user interfaces.

PROGRAMMING BY DEMONSTRATION

There are three key components in implementing a programming by demonstration system. The first is a component that *watches what the user* does. The second is a component that *infers what the user is doing*. The third is a component that automatically *performs actions on the user's behalf*.

Watching what the user does involves instrumenting the software platform on which the user is demonstrating her procedure. The instrumentation must be capable of detecting user-initiated actions. Depending on the API provided by the software platform, such actions might be low-level actions such as key presses and mouse clicks, mid-level actions such as button presses or menu selections, or high-level actions such as "launch an application".

It is important to note that the level of information returned by the instrumentation and the level required for learning by the PBD system may not match; abstraction to higher-level actions may be needed. If so, this will usually impose additional requirements on the instrumentation. For example, the system may need to translate low-level mouse click events into higher-level button presses. This requires a facility for querying and maintaining an internal representation of the on-screen widget hierarchy, in order to translate from geometric screen coordinates into widget identifiers. Another example is abstraction from lower-level actions to high-level actions such as "launch an application." To support this, the instrumentation must be able to detect and report not only user-initiated actions, but also system-initiated actions such as process creation.

Inferring what the user is doing is essential to creating a general procedure that is not tied to the particular demonstration(s) used to create it. One form of inference is generalizing parameters to the procedure, a form we call *variablization*. For example, if a reservation time of 10AM is entered when demonstrating a procedure for reserving a conference room, this should not necessarily be interpreted to mean that all future uses of that procedure will reserve for 10AM; the reservation time will typically become a procedure parameter. We assume that behavior-based customization can happen at many different levels – individual customization, customization for groups or organizations, customization for particular usage patterns. *Variablization* is critical for all of these.

A different form of inference is learning how to make decisions at choice points in the procedure. For example, a procedure to diagnose a network card might take different actions if the system is configured with a static IP address as opposed to a dynamic one. The static vs. dynamic determination can readily be made by opening a networking dialog and examining the state of a radio button that selects between the two. A user demonstrating this procedure will actually open that dialog and examine the state. The PBD system will readily learn to open the dialog, but must also learn the relevant features to examine, and the decisions to be made (i.e., the appropriate next-actions to take based on feature values) at the choice point. In the Personal Wizards system, described below, we handle *choice point inference* through the use of multiple examples.

Performing actions on the user's behalf is the final necessary component of any PBD system. When the end-user requests execution of the procedure, the PBD system must determine the sequence of actions to be taken, and execute each. In general, this will take the form of synthesizing a sequence of user actions on the existing application (or applications if the procedure is cross-application) user interface. In cases where the PBD procedure is invoked through a new control added to an existing application, this might seem to be a shortcoming – we add new interface controls that don't operate transparently. Instead, when the controls are invoked, the user sees a sequence of activations of other application controls. Although this makes the operation of the new controls idiosyncratic, it is also a strength. The user of the application can see what the new control is doing in terms of existing and understood controls, which should increase his understanding of the new control, as well as his trust that it is performing its functions correctly.

An important consideration in designing a PBD system is how to give the user a view of the generated procedure. This is important not only to edit the procedure (assuming that the PBD system supports that), but also to give users the opportunity to develop trust in the generated procedure by inspecting it. In addition, control over execution of the generated procedure can be important, such as an ability to see what the procedure will do next and to manually specify whether or not the procedure will be allowed to continue. Although procedures can often be automatic, this is not universally true; many procedures require human judgment, particularly before taking steps that are irreversible or have other strong consequences.

In the next section, we discuss the Personal Wizards system for programming by demonstration on the desktop.

PERSONAL WIZARDS

Personal Wizards is desktop programming by demonstration system that runs on Microsoft Windows platforms (note that Personal Wizards is a slightly enhanced version of the Sheepdog system described in [Lau04]). Personal Wizards differs from prior PBD systems in its ability to learn from multiple demonstrations of the same procedure. By utilizing a number of examples that follow different branches in the procedure (e.g., running on different versions of the operating system), we are able to infer a rich procedure model containing choice points.

Based on a belief that procedures should not be black boxes, and that many procedures cannot simply be automated, Personal Wizards procedures are inspectable and allow the end user to control their execution by manually stepping through them. These features will be described in more detail in the next section.

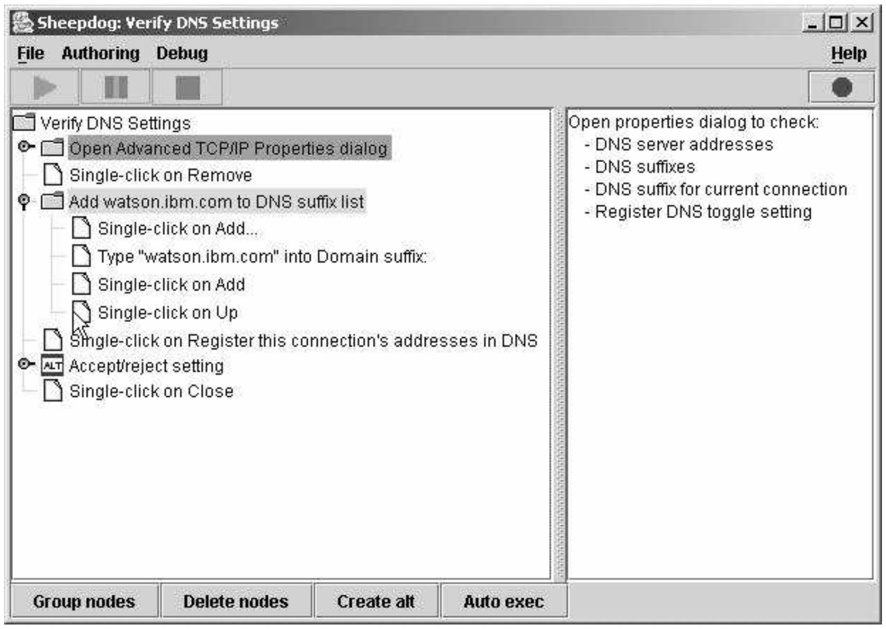
User Interface

There are three modes or roles in which a user may interact with the Personal Wizards system. A single unified interface (figure 1) supports all three modes. The first mode is *record mode*. Pressing the *record* button in the user interface causes Personal Wizards to begin capturing user and system actions. All actions are captured and saved to file with the exception of actions on the Personal Wizards interface itself. As each user action is detected, it is added to a list of actions in the Personal Wizards interface. These are currently reported as low-level actions with minimal abstraction to identify the target component for the operation, for example, "Double click on My Computer".

Once one or more demonstrations have been recorded, the user may edit the procedure in *authoring mode*. Authoring, which is an optional step, makes the procedure more understandable by adding hierarchical structure, as well as specifies which subprocedures are to be automated and which are to require manual initiation.

The final interface mode is *play mode*. Here the procedure is interactively executed, with manual control over execution of steps or subprocedures. Personal Wizards highlights each onscreen control (button, icon, etc) prior to activating it, and advances an execution cursor within the procedure display, highlighting the step about to be executed. A play button is used to execute the current step or subprocedure, and to advance to the next.

In addition to the automated execution, Personal Wizards also supports a limited form of *mixed initiative* [Wolfman01]. If the user manually performs operations in the application interface (such as pressing a button or selecting from a menu) that were seen during any of the recording sessions, Personal Wizards will recognize the action, predict a next step for the procedure, and highlight in the interface the action associated with that step, as well



Verify DNS Settings
- Open Advanced TCP/IP Properties dialog
  - Single-click on Remove
- Add watson.ibm.com to DNS suffix list
  - Single-click on Add...
  - Type "watson.ibm.com" into Domain suffix:
  - Single-click on Add
  - Single-click on Up
  - Single-click on Register this connection's addresses in DNS
- Accept/reject setting
  - Single-click on Close

Open properties dialog to check:
- DNS server addresses
- DNS suffixes
- DNS suffix for current connection
- Register DNS toggle setting

Figure 1. Personal Wizards user interface showing a networking procedure during playback

as highlighting the control onscreen associated with the action. This allows Personal Wizards to function as a sort of tutorial system – at each step that the user performs, if Personal Wizards recognizes that place in the procedure, it will inform the user of the probable next action, allowing her to either continue on manually (accepting or ignoring the recommendation), or press the play button to resume automated execution. This facility also allows the user to diverge from the procedure path, either to execute alternative actions, or to engage in a different activity (such as going off and reading email); when the user is "back on track", Personal Wizards will recognize that, and inform the user appropriately. In the future, we plan to extend this mixed initiative processing to allow the user to enter recording mode dynamically so that new pathways through a procedure can be captured.

Usage Scenarios

We expect Personal Wizards to make its first impact as a system administrator support tool, or for use in deskside support. In these domains there are numerous well-defined tasks – moving file systems, installing software patches, correcting particular error conditions, etc – that we believe amenable to programming by demonstration. We envision well-trained experts creating these procedures, either by recording activities while troubleshooting in the field and latter assembling and authoring the procedure, or by explicitly recording/authoring a procedure for which wide use is anticipated. Dissemination of the executable procedures can be through websites, or email attachments. Note that Personal Wizards currently requires a manual installation on each client platform; we expect to have a web-based install within the near future.

A second category of use for Personal Wizards will be in automating business processes. We believe that many business processes can be thought of as "well worn paths" through fairly complicated user interfaces, often involving more than one software package. Personal Wizards will allow the capture of "best practices" in performing these common operations, and allow an organization to make them available on corporate websites, or even as additional controls within particular applications. This usage category can be thought of as organizational customization.

A third category will be personal customization. Currently we envision the end-user explicitly recording procedures (and authoring when desirable) and explicitly invoking them. Although we can imagine using inference techniques to determine when Personal Wizards should automatically record and/or suggest playback, this is an open research area, and great care needs to exercised to not repeat the well-known "paperclip" debacle.

A key to enabling all of these usage scenarios will be creating searchable procedure repositories. Although keyword specification by the authors will go partway in making procedure repositories useful, automated labeling and search will be required to support large repositories. This is an area for future research, which we will be actively pursuing.

Customizing a User Interface using Personal Wizards

We have devised several ways in which Personal Wizards might be used to customize existing user interfaces. Although these are currently unimplemented, we will give a brief overview of some of the possible approaches.

The first possibility is to add controls to existing applications. We call this *scaffolding*, since the existing application is being used as a scaffold to contain a set of new user-defined controls. There are two ways in which scaffolding could be implemented.

The first is for individual applications to be extended to permit the user to define new controls and bind PBD procedures to them. A number of existing applications allow the user to extend the control set. An excellent example is the Eclipse platform [Eclipse04] which allows a developer to define plugins, and through an XML specification define new interface extensions such as toolbar buttons and menu items. Although this is the most reliable and easy-to-use mechanism, unless an API is already provided, it requires alteration to the application source for each application that is to have new controls installed. Note also that such extensions would require either an API for the application to communicate with the Personal Wizards process, or repackaging Personal Wizards as a component.

A more generic approach is to modify an application's control set using operating system API calls. For example, Windows provides API calls that allow a program to add a button to a toolbar even within a different process. A callback is registered with that control. This would allow a Personal Wizards process to install controls in a variety of application user interfaces, with the appropriate Personal Wizards procedure being executed when each is activated.

A related idea is one we call *control skinning*. This is based on the commonly known notion of skinning, in which an application provides an API that allows an end-user to remap the visual appearance of individual elements (such as buttons) within a user interface. We propose extending the idea to remapping existing controls within an interface to a new and different functionality.

As with scaffolding there are two possible approaches to this. The first is to modify existing applications to permit this remapping. A control, once remapped, would invoke a PBD procedure, rather than the original function.

The other approach is to implement control skinning through the operating system. This is a bit trickier than scaffolding, since the Personal Wizards process will need to intercept events that are intended for the application. If the operating system allows insertion of a listener (callback) prior to any application listeners, as well as removal of the event from the event queue, we have the necessary mechanisms to accomplish this. Although Windows hooks allows registration of event listeners, we do not yet know if it is possible to ensure that our listeners are invoked prior to the application listeners.

An alternate operating system-based approach is to physically layer controls on top of the existing controls. Transparent windows can be placed directly on top of the control to be remapped. We call such overlays *appliqués*. Note that appliqués need not be transparent, they can be used to alter the physical appearance of the application controls (this is traditional skinning, except done outside of the application) in addition to altering functionality. Since the appliqué is above the application window in z-ordering, it receives mouse events, rather than the application. This begs the question of hotkey handling, however, which must be implemented using the previously described mechanism. An additional complication is how to handle window move and resize events. We must be able to register for these, and make the appropriate changes to the appliqué size and position to keep it registered with the application window.

Note that in the extreme case we can simply use screen capture techniques to create a new window that emulates the entire application window. Essentially we have created an application proxy, which processes remapped controls (by invoking the appropriate Personal Wizards procedure), and passes events along to the application process as appropriate. The original application window is minimized, and as new application windows are created, proxies are generated. We envision applying the ideas of scaffolding and control skinning in conjunction with PBD to develop a PBD-based user interface customization toolkit. In addition to the recording and authoring components currently within Personal Wizards, the interface would provide the user with the ability to add new controls to existing interfaces, as well as binding new functionality, and optionally new appearance to existing controls. We are currently beginning to prototype such a customization toolkit.

REFERENCES

1. [Cypher93] Cypher, A., Ed. Watch What I Do -- Programming by Demonstration. The MIT Press, Cambridge, MA 02142, 1993.
2. [Eclipse04] www.eclipse.org
3. [Lau04] Lau, T, Bergman, L., Castelli, V., Oblinger, D., Sheepdog: Learning Procedures for Technical Support, IUI 2004.
4. [Lieberman01] H. Lieberman, Ed. Your Wish is My Command -- Programming by Example. Morgan Kaufmann Publishers, 2001.
5. [Wolfman01] Wolfman, S. A., Lau, T. A., Domingos, P., Weld, D. S., "Mixed initiative interfaces for learning tasks: SMARTedit talks back", IUI2001, pp. 167-174.