Developing a Practical Programming by Demonstration Tool

Gordon W. Paynter and Ian H. Witten

Department of Computer Science
The University of Waikato
Private Bag 3105,
Hamilton,
New Zealand

Email: gwp@cs.waikato.ac.nz, ihw@cs.waikato.ac.nz

Fax: +64 7 838 4155

***Principal contact**: Gordon W. Paynter*

***Category**: full paper*

Developing a Practical Programming by Demonstration Tool

Gordon W. Paynter and Ian H. Witten
Department of Computer Science
The University of Waikato
New Zealand
{gwp,ihw}@cs.waikato.ac.nz

Abstract

Many iterative tasks in direct manipulation interfaces cannot be automated with standard application tools, forcing users to repeat the same interface actions again and again. We describe a domain-independent programming by demonstration system that learns iterative tasks in a range of widely-used applications on a popular computer platform. An evaluation showed that users are capable of using the agent to automate iterative tasks, and will choose to do so in many circumstances. It also found many shortcomings, which were corrected in a new version of the interface. This paper recounts the design process, the first interface, the evaluation, and consequent revisions.

1. Introduction

Computers excel at performing repetitive tasks, and can spare us hours of mindless drudgery. But it is not always possible to make an interactive computer application repeat a particular task. Some applications provide aggregation techniques like multiple selection, but these are only suitable for certain tasks. An experienced programmer might write a program or script to automate iteration, but this lies beyond the abilities of most end-users, who are often forced to perform the same interface actions over and over again.

Programming by demonstration (PBD) is a technique that allows end users to solve iteration problems [1]. The user can teach the system to perform a task by demonstrating what they want done. PBD systems can learn programs from users who have no programming knowledge—they need to know nothing more than how to perform the task.

Demonstrational techniques have been used for a wide range of purposes: teaching; generating code; building applications; scheduling; creating macros, scripts and commands [1]. Most systems for automating repetition are implemented in research environments and are not amenable to evaluation by real users. Those that do use existing software are either tied to specific applications [3,4] or operate at a very low level.

Our work is motivated by several questions that previous projects have left unanswered. Can PBD be added to commercial applications (like Microsoft Word or Excel)? If users are presented with a PBD system in a familiar environment, can they use it? Will they choose to? Will they choose it over alternative techniques?

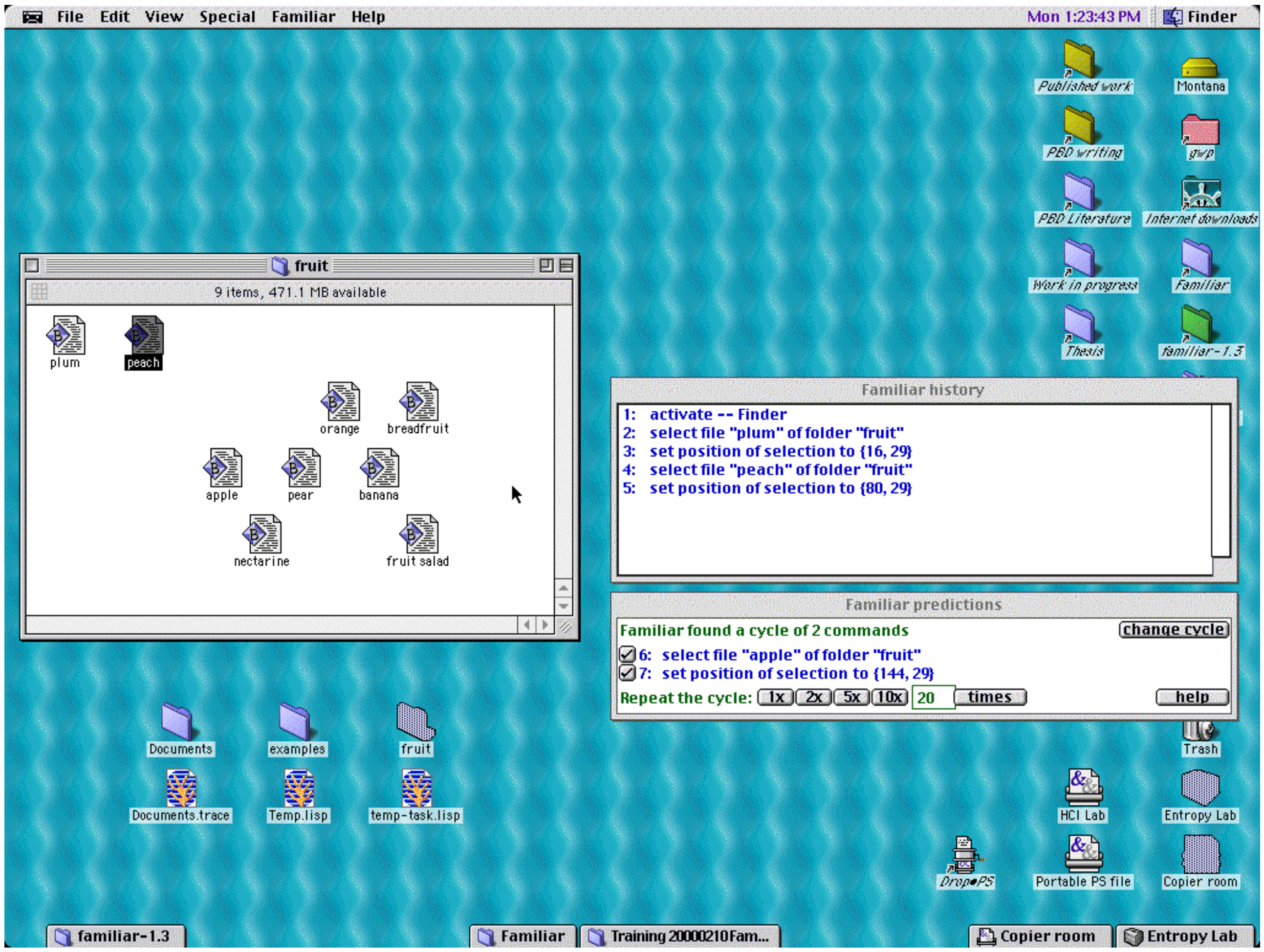
To answer these questions, we have designed and implemented Familiar, an application-independent PBD system for the Macintosh computer, and evaluated it by asking users to automate iterative tasks. We found that end-users, including non-programmers, can use the interface to automate iteration, though they often prefer alternative techniques when they are available [5].

This paper focuses on another outcome of the evaluation: the subjects discovered several problems that led to a redesign of the interface with the aim of making it easier to understand and control. The new interface predicts more aggressively, but its predictions can be explained and corrected through the enhanced interface.

2. Iteration and demonstration

Iterative tasks are those that the user completes by repeating a series of interface actions without interleaving actions from other tasks. For example, a user who works through the records in a database, entering a value in a field of each and immediately moving to the next, is manually performing an iterative task. Our aim is to help end users automate these tasks in direct-manipulation graphical user interfaces.

PBD is an end–user programming technique that can help solve iteration problems. A PBD interface allows the user to "program" the computer by giving demonstrations. The process can be likened to teaching: the user presents specific examples of what they want done, and the computer learns a general strategy for performing the task that it can apply to new examples. To create a conventional program, the user must describe the entire task in advance, in flawless detail, in an abstract form, in a foreign environment. To create a program by demonstration, a user simply needs to perform the task within a conventional user interface.



1: activate -- Finder
2: select file "plum" of folder "fruit"
3: set position of selection to {16, 29}
4: select file "peach" of folder "fruit"
5: set position of selection to {80, 29}

Familiar found a cycle of 2 commands

☑ 6: select file "apple" of folder "fruit"
☑ 7: set position of selection to {144, 29}

Repeat the cycle: 1x 2x 5x 10x 20 times

Figure 1 Using Familiar 1.3 to arrange icons

Familiar is a PBD system for solving iterative tasks. It is unique in that it works with a range of existing, unmodified applications on a commercial platform—the Apple Macintosh—rather than with specially-designed research prototypes. It is neither tied to specific applications, nor restricted to one application at a time. It works out-of-the-box with new scriptable applications, relieving the developer of any need to compile domain knowledge. Familiar is thus a practical PBD system.

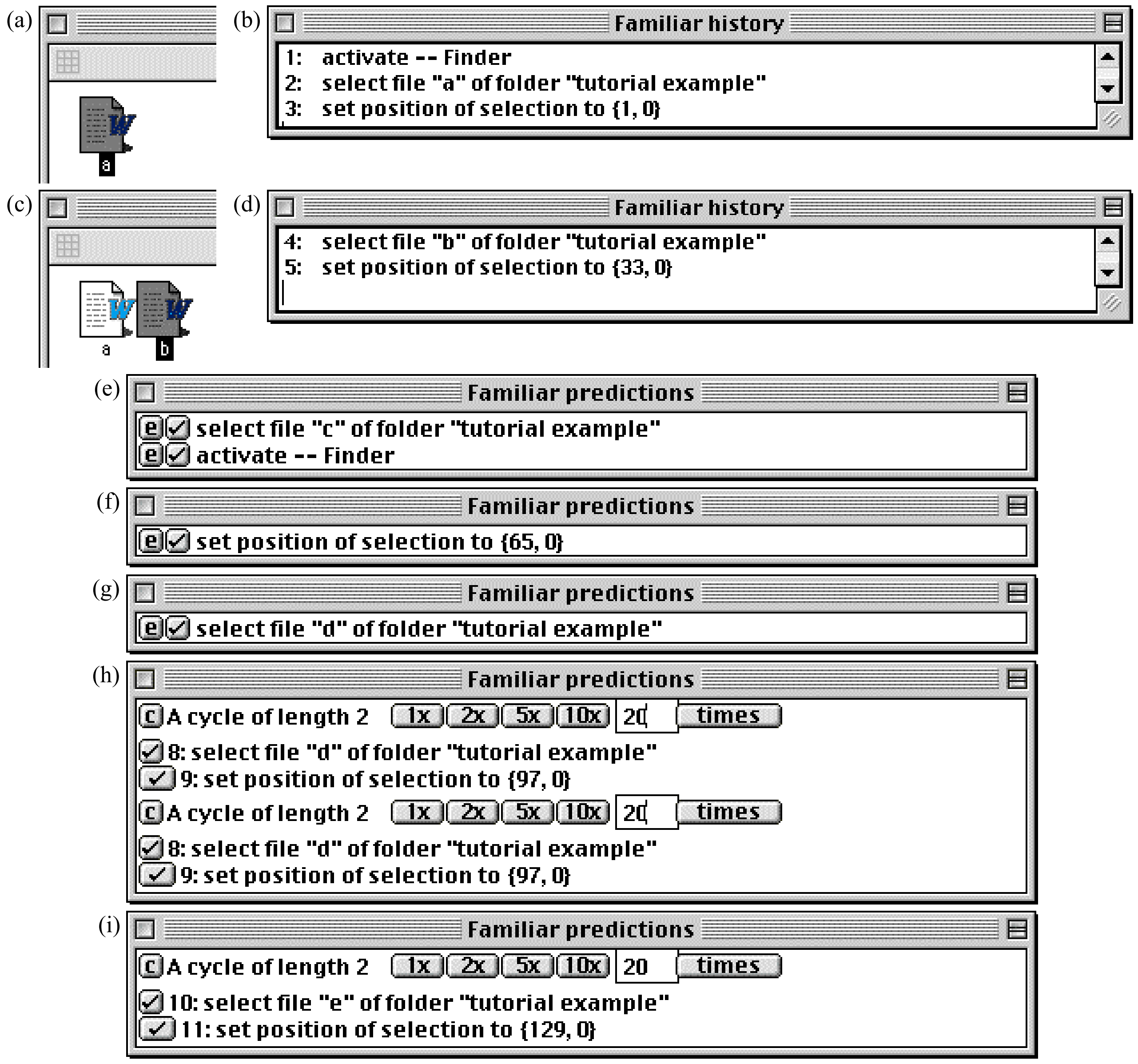
Figure 1 shows the screen of a user who is completing a simple task. Familiar has been invoked from the *Familiar* menu, which is present in the menubar of every application (next to the ubiquitous *Help* menu). A flashing tape-recorder icon in the top left corner of the screen indicates that the user's actions are being observed. This user is arranging the icons in a folder into a row. Two files, named *plum* and *pear*, have been moved to their new positions and are visible in the window to the left of the screen. Two other windows appear to the right. The first, titled *Familiar history*, contains a textual description of the actions the user has recently taken. The second, *Familiar predictions*, shows what Familiar thinks the user will do next. Familiar is a stand-alone application, and its windows remain in the background until the user selects one, bringing it to the foreground. The agent can then be interrogated and instructed, as described below.

Figure 1 shows the most recent form of the interface, Familiar 1.3. This paper explains how it was developed.

3. Designing the Familiar 1.0 interface

Many PBD systems can best be described as "feature-oriented": they are created to exhibit and evaluate some unique interaction or inferencing technique, a purpose they usually fulfil. Unfortunately, these systems are ill-suited to real users: they are unpolished, unfamiliar, and lack many essential application features.

In contrast, Familiar is designed for end users. We considered their motivations, abilities, and attitudes, and formulated four guidelines for constructing user-centric PBD systems: (a) the use of existing applications, (b) simplicity, (c) minimising user actions, and (d) educating the user [5]. The extent to which these goals can be satisfied is bounded by their contradictory nature and by the platform. Nevertheless, the resulting system, Familiar 1.0, is more accessible than other PBD tools because it works in standard application programs.



```
1:  activate -- Finder
2:  select file "a" of folder "tutorial example"
3:  set position of selection to {1, 0}
```

```
4:  select file "b" of folder "tutorial example"
5:  set position of selection to {33, 0}
```

```
select file "c" of folder "tutorial example"
activate -- Finder
```

```
set position of selection to {65, 0}
```

```
select file "d" of folder "tutorial example"
```

```
A cycle of length 2   20 times
8: select file "d" of folder "tutorial example"
9: set position of selection to {97, 0}
A cycle of length 2   20 times
8: select file "d" of folder "tutorial example"
9: set position of selection to {97, 0}
```

```
A cycle of length 2   20 times
10: select file "e" of folder "tutorial example"
11: set position of selection to {129, 0}
```

Figure 2 Arranging files with Familiar 1.0

3.1. Teaching a simple task

Users are trained to use Familiar with a tutorial that asks them to perform a simple task: open a folder of 26 files, and arrange them in a row. Familiar is capable of considerably more sophisticated inferencing [5]: the point of this example is to explain the user interface.

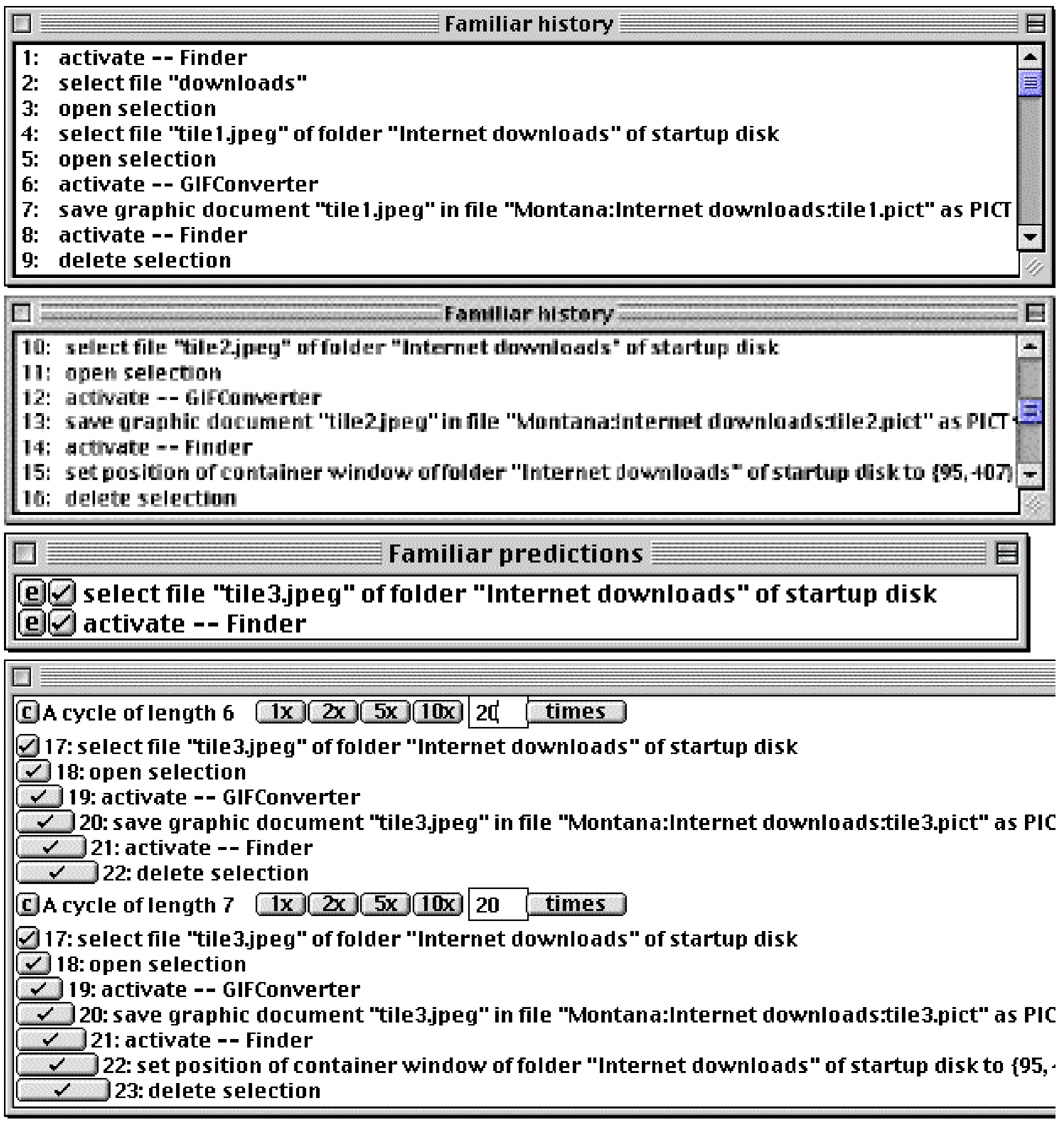
To begin any task, the user asks the agent to start observing their demonstration by selecting *Begin recording* from the *Familiar* menu, which is available in every application (and visible in Figure 1). They then proceed to demonstrate the task in the standard user interface of the appropriate application—or applications, for Familiar spans applications.

Next, the user selects the *tutorial example* folder in the Finder and puts the file named *a* in the top left corner of the window, as shown in Figure 2a. As users work, a window labelled *Familiar history* appears and logs each action. Actions are described in AppleScript, an English-like scripting language built into the Macintosh operating system and applications. At this point the *history* window contains three commands (Figure 2b) corresponding to the high-level events the user has performed: activating the Finder, selecting file *a*, and positioning that selection.

Continuing the demonstration, the user places file *b* beside file *a* (Figure 2c). Again, the commands are recorded in the history window (Figure 2d).

3.2. Familiar performs the task

At this point in the demonstration, Familiar has recorded two iterations of a task and begins making predictions. A new window, titled *Familiar predictions*, is created: it displays two predictions of the next event (Figure 2e).

(a)
(b)
(c)
(d)

Familiar history

```
1:  activate -- Finder
2:  select file "downloads"
3:  open selection
4:  select file "tile1.jpeg" of folder "Internet downloads" of startup disk
5:  open selection
6:  activate -- GIFConverter
7:  save graphic document "tile1.jpeg" in file "Montana:Internet downloads:tile1.pict" as PICT
8:  activate -- Finder
9:  delete selection
```

Familiar history

```
10:  select file "tile2.jpeg" of folder "Internet downloads" of startup disk
11:  open selection
12:  activate -- GIFConverter
13:  save graphic document "tile2.jpeg" in file "Montana:Internet downloads:tile2.pict" as PICT
14:  activate -- Finder
15:  set position of container window of folder "Internet downloads" of startup disk to {95, 407}
16:  delete selection
```

Familiar predictions

```
e ✓ select file "tile3.jpeg" of folder "Internet downloads" of startup disk
e ✓ activate -- Finder
```

```
c A cycle of length 6   1x 2x 5x 10x  20  times
✓ 17: select file "tile3.jpeg" of folder "Internet downloads" of startup disk
✓ 18: open selection
✓ 19: activate -- GIFConverter
✓ 20: save graphic document "tile3.jpeg" in file "Montana:Internet downloads:tile3.pict" as PIC
✓ 21: activate -- Finder
✓ 22: delete selection
c A cycle of length 7   1x 2x 5x 10x  20  times
✓ 17: select file "tile3.jpeg" of folder "Internet downloads" of startup disk
✓ 18: open selection
✓ 19: activate -- GIFConverter
✓ 20: save graphic document "tile3.jpeg" in file "Montana:Internet downloads:tile3.pict" as PIC
✓ 21: activate -- Finder
✓ 22: set position of container window of folder "Internet downloads" of startup disk to {95,
✓ 23: delete selection
```

Figure 3 Changing image formats with Familiar 1.0

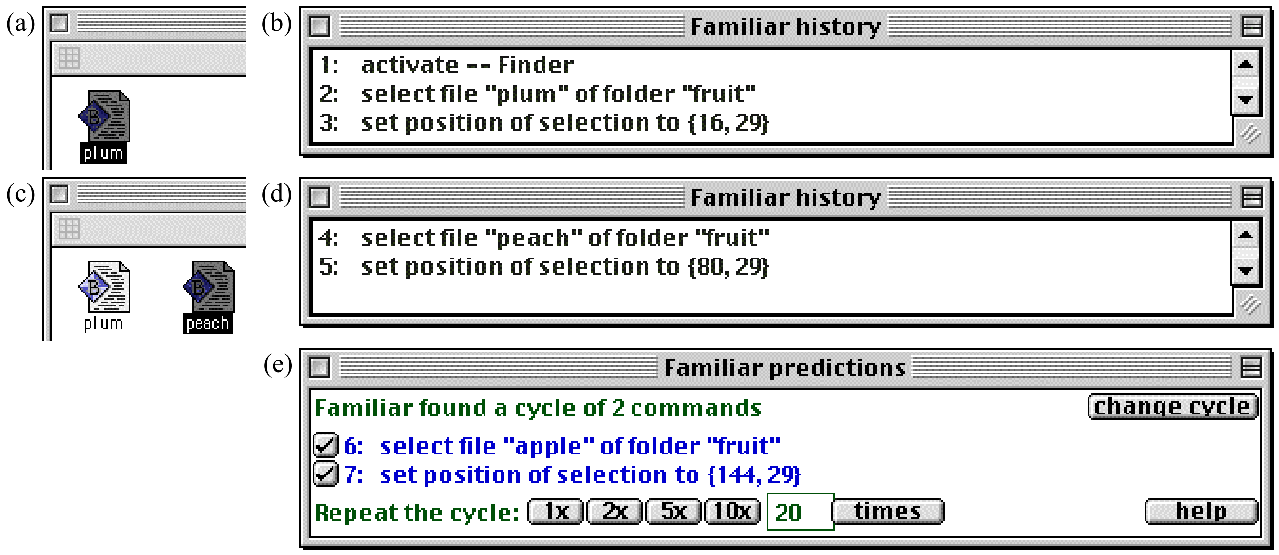
The first is correct—the next action the user intends to take is *select file "c"*. The second is incorrect; it was generated by assuming that the *activate* command is a necessary step, but that the user, through oversight, forgot to include it in the second demonstration. In fact, the *activate* command does nothing in this context, but Familiar knows only the syntax of the commands, not their effect.

Familiar can be asked to execute a prediction by clicking either the text of the command or the tick button beside it. When the user clicks the first command in Figure 2e, four things happen. First, the text of the command changes colour, indicating that it has been sent to the application. Second, the application carries out the command, and the user sees file *c* being selected in the Finder. Third, the command is recorded by Familiar and added to the *history* window, just as any user command would be. Finally, the *prediction* window is updated with a prediction of the next command.

The next prediction suggests that the user sets the *position* of the *selection* (Figure 2f). The user accepts this prediction by clicking on it, and watches as the file is moved in the folder window. A new prediction appears: that the user will *select file "d"* (Figure 2g).

The user does want to select *d*, and could easily tell Familiar to carry out this command, but it is inefficient to ask the agent to execute one step at a time—it would probably be faster for the user to move the files by hand! Instead of accepting this prediction, the user clicks on the *expand* button (labelled "e"), and Familiar attempts to predict complete iterations of the task.

Familiar predicts two cycles (Figure 2h). Both appear to be the same—a *select* command followed by a *set* command—so the user ignores the second cycle. The topmost cycle correctly anticipates that the user's next action (step 8) will be to *select file "d"* and that the one after that will be to set the *position* of the *selection* to {97, 0} (step 9). There are two ways to make Familiar perform a complete cycle of the task. One is to click on



1: activate -- Finder
2: select file "plum" of folder "fruit"
3: set position of selection to {16, 29}

4: select file "peach" of folder "fruit"
5: set position of selection to {80, 29}

Familiar found a cycle of 2 commands
✓ 6: select file "apple" of folder "fruit"
✓ 7: set position of selection to {144, 29}
Repeat the cycle: 1x 2x 5x 10x 20 times

Figure 4 Arranging files with Familiar 1.3

the last command in the cycle (in this case, step 9). Clicking on any command will execute the commands up to and including that one. The other is to click on the *one times* button (labelled "1x").

In this example, the user executes the cycle by clicking the *one times* button, and the commands are executed in sequence. Each command is recoloured before it is sent to the application so that the user can see the agent's progress, and the other Familiar windows are updated as usual. When the two commands have been executed, Familiar predicts the next iteration of the cycle (Figure 2i).

The user is now confident that Familiar has learned the task, and can use the controls in the prediction window to make Familiar perform more than one iteration at a time. The user can press the *two times* ("2x"), *five times* ("5x"), or *ten times* ("10x") buttons to execute the corresponding number of iterations; or type an arbitrary number of repetitions in the field provided and press the button marked *times*. When the user asks for more than one iteration, this field is dynamically updated to display the number of cycles that remain.

The user must consider the termination conditions of the task. Familiar will run for the specified number of iterations, or until it cannot predict the next event, or until an error is generated by one of the commands it executes. Many tasks have a natural limit—in this case the number of files in the folder—that, when exceeded, automatically trigger an error. In some situations Familiar does not stop where the user might expect it to—for example, when Familiar has filled the top row of the window with files, it keeps placing them on the same row outside the visible area of the window.

3.3. Another example

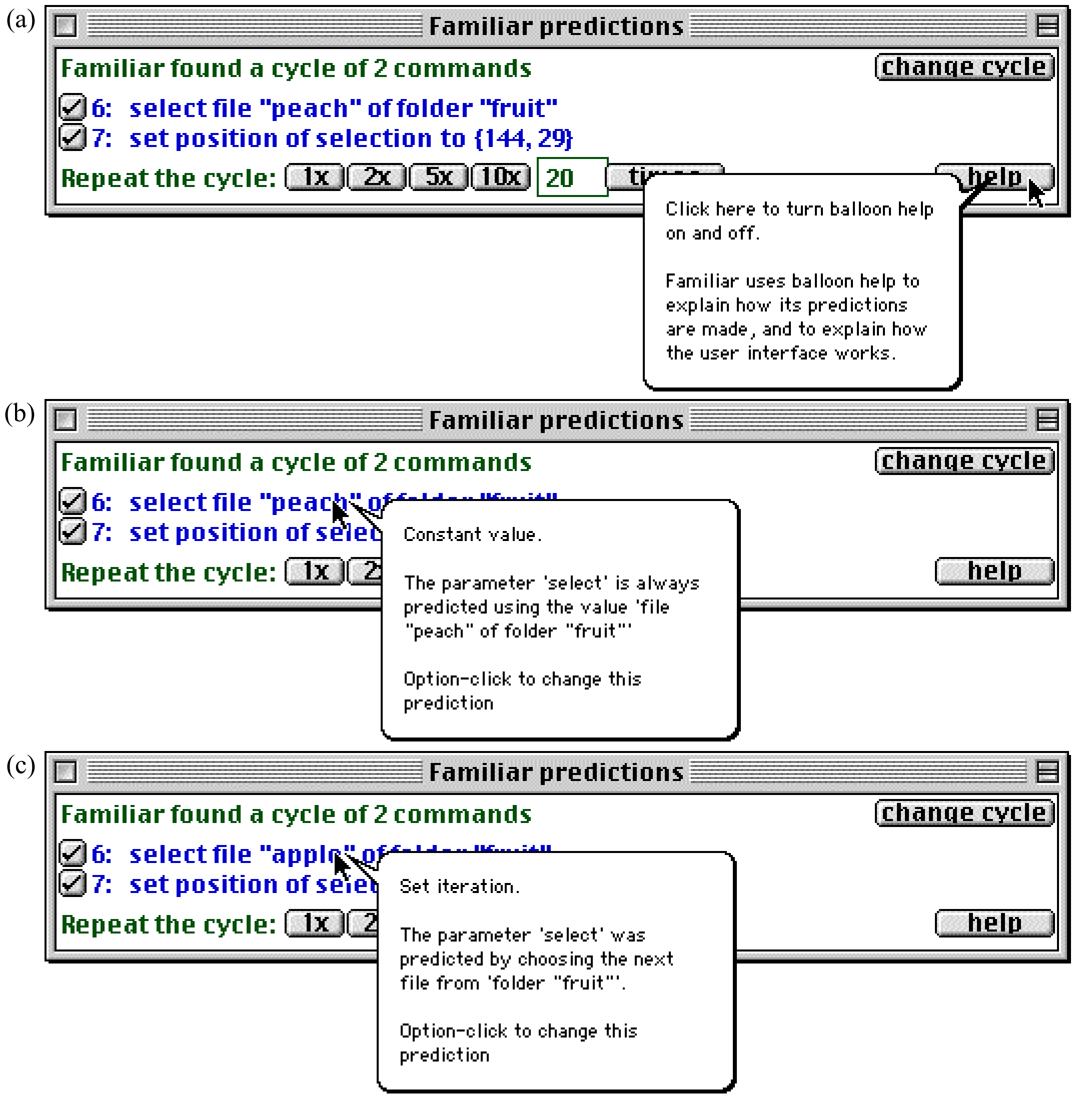
Complex tasks may involve many applications and imperfect demonstrations. Figure 3 shows a user changing the format of a set of image files. The *history* window is shown after the first (Figure 3a) and second (Figure 3b) iterations have been demonstrated. The first three events of Figure 3a initialise the environment and are not part of the iterative loop. Event 15 is a singular noise event generated when the user shifted a window to get a better view, and should not be repeated in future iterations. Figure 3c shows that Familiar correctly predicts the next event. When it is expanded in Figure 3d we see a cycle of six events that correctly predicts the next iteration, and another of seven events that includes an extra step (event 22) corresponding to event 15 of Figure 3b.

4. Evaluating Familiar 1.0

The Familiar 1.0 interface was evaluated by a group of ten end users. The evaluation had three objectives. The first was to test the hypothesis that users are capable of automating iterative tasks with PBD. The second was to find out whether users will choose to use Familiar when other automation tools are available. The third was to gather feedback from end users about the interface.

The ten subjects were asked to solve two iterative tasks, then introduced to Familiar and asked to complete much larger variants of the tasks. The tasks were performed in four applications: the Finder, the Excel spreadsheet, the Fetch FTP client, and the GIFConverter image manipulation program. Subjects' ability to use PBD was tested by observing their use of Familiar. All were able to do so with little training. Tool choice was assessed by observing whether the subjects chose to invoke Familiar or existing automation facilities based on multiple selection. Most subjects elected to use the existing tools when they were available. These aspects of the evaluation are discussed in [5].

All subjects were observed using Familiar, and were asked to comment on it. The subject's opinions and their


(a)

Familiar predictions

Familiar found a cycle of 2 commands

6: select file "peach" of folder "fruit"

7: set position of selection to {144, 29}

Repeat the cycle: 20

Click here to turn balloon help on and off.

Familiar uses balloon help to explain how its predictions are made, and to explain how the user interface works.

(b)

Familiar predictions

Familiar found a cycle of 2 commands

6: select file "peach" of folder "fruit"

7: set position of selec

Repeat the cycle:

Constant value.

The parameter 'select' is always predicted using the value 'file "peach" of folder "fruit"'

Option-click to change this prediction

(c)

Familiar predictions

Familiar found a cycle of 2 commands

6: select file "apple" of folder "fruit"

7: set position of selec

Repeat the cycle:

Set iteration.

The parameter 'select' was predicted by choosing the next file from 'folder "fruit"'.

Option-click to change this prediction

Figure 5 Using ballon help to get explanations of predictions in Familiar 1.3

observed behaviour identified several problems with the interface. These problems are described below, with potential solutions to each.

Problem 1 The most common complaint was that Familiar is slow. Its speed is restricted by AppleScript, which is a slow language, and by the user's hardware.

Solution Although beyond Familiar's control, this problem is ameliorated by faster hardware and later software versions. The evaluation was performed on a 200MHz Power Macintosh running AppleScript 1.1.2 (based on 68000 machine code). A newer 400MHz G3 Power Macintosh using AppleScript 1.2 (native PowerPC code) is dramatically faster.

Problem 2 Familiar makes predictions that do not match the user's mental intent.

Solution The interface must explain its predictions.

Problem 3 Familiar occasionally makes poor predictions in the face of multiple counterexamples, or takes too long to make predictions. In many cases the correct behaviour is obvious to the subject, and Familiar is able to make the correct prediction but lacks data.

Solution The user should be able to explicitly reject incorrect predictions and request new ones.

Problem 4 Some users had difficulty recognising and changing between "step" mode and "cycle" mode.

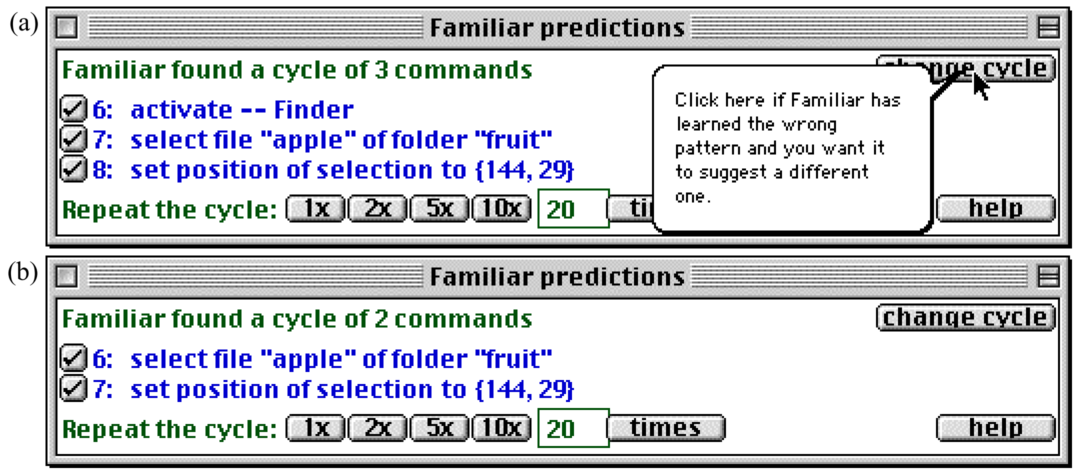
Solution Eliminate "step" mode.

Problem 5 Having two cycles shown at the same time is confusing, particularly if they appear the same.

Solution Display a single cycle and allow the user to change it.

Problem 6 Some subjects identified problems with Familiar's handling of termination conditions. It can be difficult to calculate the required number of iterations.

Solution The interface should suggest termination conditions based on the current data (e.g. number of files in a folder).


(a)

Familiar predictions

Familiar found a cycle of 3 commands

✓ 6: activate -- Finder

✓ 7: select file "apple" of folder "fruit"

✓ 8: set position of selection to {144, 29}

Repeat the cycle: 1x 2x 5x 10x 20

Click here if Familiar has learned the wrong pattern and you want it to suggest a different one.

(b)

Familiar predictions

Familiar found a cycle of 2 commands

✓ 6: select file "apple" of folder "fruit"

✓ 7: set position of selection to {144, 29}

Repeat the cycle: 1x 2x 5x 10x 20 times

Figure 6 Using the change cycle button to correct a prediction in Familiar 1.3

Problem 7 Some subjects commented that Familiar was unsuitable for short iterations because it was too slow to learn changes in patterns.

Solution Reimplement the parameter extrapolation algorithm.

Problem 8 The subjects were asked whether they felt they understood AppleScript. Eight claimed they did. The other two were inexperienced computer users, and one thought he would learn in time. Four of the subjects had queried the terminology used by Excel, and later acknowledged it had initially caused them difficulties.

Solution AppleScript is beyond our control because each application's developer defines its terminology. Elsewhere we have suggested guidelines for creating "PBD-friendly" implementations [5].

5. The Familiar 1.3 interface

The solutions recommended above, except for Problem 6, are incorporated into Familiar 1.3. To illustrate the differences between the two interfaces, we describe example tasks like those of Section 3.

5.1. Arranging files

To open a folder and arrange its files as in the first example above, the user, having invoked *Begin recording* from the *Familiar* menu, selects the appropriate folder and moves a convenient file, *plum*, to the top left of the folder window (Figure 4a). These actions are recorded and displayed in the history window (Figure 4b). The user continues the demonstration by moving file *peach* (Figure 4c); again their actions are recorded and displayed (Figure 4d). So far, the interface is unchanged from version 1.0.

In the earlier interface, Familiar would now predict the next step in the cycle (Figure 2e). However, the new interface does not predict single steps; instead it displays its best prediction of the entire next cycle. In this case, it predicts both step 6 and step 7 correctly (Figure 4e).

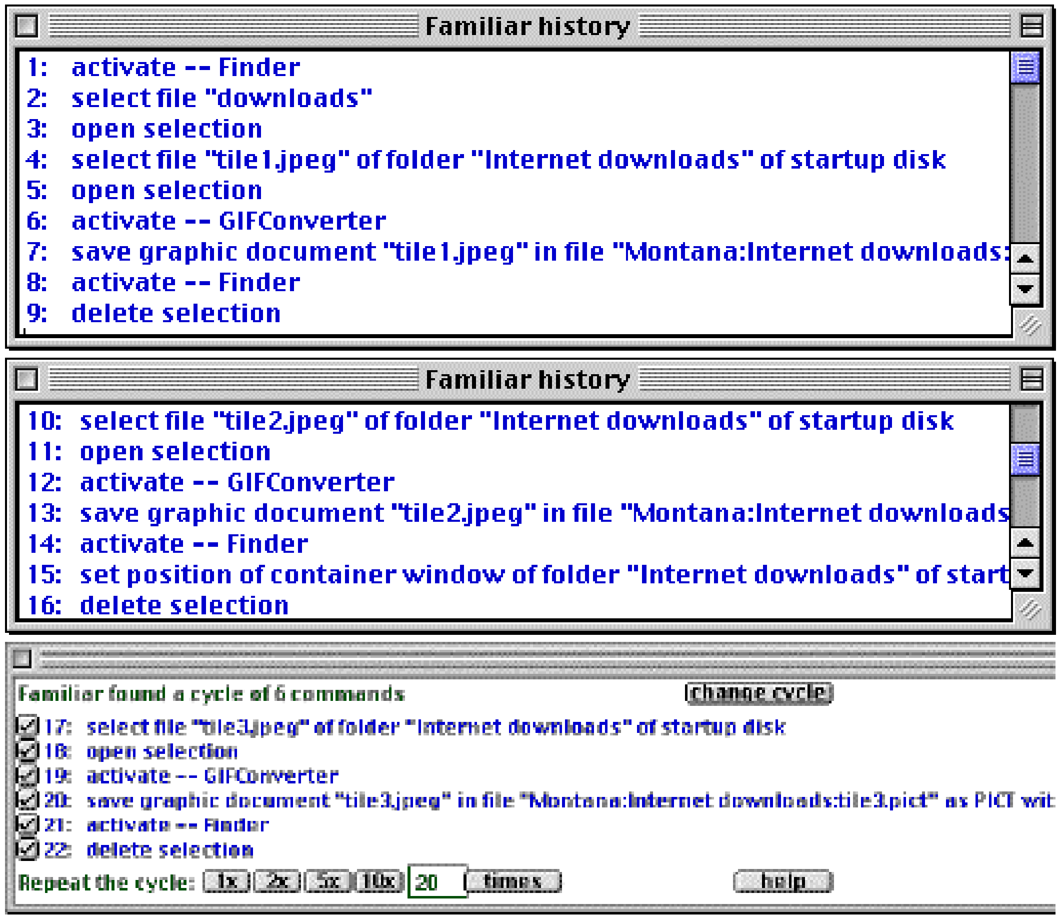
Figure 1 shows the entire screen as it appears after the user has demonstrated the first two examples. The controls at the base of the predictions window operate just like those in the version 1.0 interface, and the user can complete the task by requesting the appropriate number of iterations.

5.2. When Familiar makes errors

In the original interface, the only way to correct one of Familiar's predictions was to demonstrate another example in the application interface. This option is still available in the new interface, but several new features for communicating with Familiar have been added.

The *help* button gives feedback explaining how predictions are made. The iterative pattern predicted in Figure 5a is consistent with the two demonstrations of the task above (Figure 4a–d), but the parameter of the *select* command (event 6) has not been extrapolated correctly: Familiar has predicted that the user will select *peach*, but the user already moved this file (events 4 and 5) and wants to move a new one. To find out why the agent made the erroneous prediction, the user clicks *help*, which activates the Macintosh Balloon Help feature (Figure 5a). The user is concerned that the *select* parameter is incorrect, so moves the mouse pointer over that prediction. A help balloon explains that Familiar has reasoned that the user is selecting the same file in every iteration, and that the prediction can be changed with option-click (Figure 5b). The user tells Familiar to change the prediction, and *peach* is immediately replaced by *apple* (Figure 5c). The help balloon explains that the new prediction is made by assuming that the user is iterating over all the *file* objects in *folder "fruit"*. The agent's reasoning—and thus its prediction—is correct, and the task can now be completed.

The *change cycle* button is used to reject the current iterative pattern and display another. Suppose that after the user demonstrated the first two examples of the task (Figure 4a–d), Familiar deemed the *activate* command to be important and predicted that each cycle was composed of three events, *activate*, *set*, and *select*, as


(a)

Familiar history

```
1:  activate -- Finder
2:  select file "downloads"
3:  open selection
4:  select file "tile1.jpeg" of folder "Internet downloads" of startup disk
5:  open selection
6:  activate -- GIFConverter
7:  save graphic document "tile1.jpeg" in file "Montana:Internet downloads:
8:  activate -- Finder
9:  delete selection
```

(b)

Familiar history

```
10:  select file "tile2.jpeg" of folder "Internet downloads" of startup disk
11:  open selection
12:  activate -- GIFConverter
13:  save graphic document "tile2.jpeg" in file "Montana:Internet downloads
14:  activate -- Finder
15:  set position of container window of folder "Internet downloads" of start
16:  delete selection
```

(c)

Familiar found a cycle of 6 commands

```
☑ 17:  select file "tile3.jpeg" of folder "Internet downloads" of startup disk
☑ 18:  open selection
☑ 19:  activate -- GIFConverter
☑ 20:  save graphic document "tile3.jpeg" in file "Montana:Internet downloads:tile3.pict" as PICT wit
☑ 21:  activate -- Finder
☑ 22:  delete selection
```

Repeat the cycle: 1x 2x 5x 10x 20 times

Figure 7 Changing JPEG images to PICT images with Familiar 1.3

shown in Figure 6a. To remedy this error, the user presses *change cycle*, and Familiar replaces the three-event cycle with the correct two-event one (Figure 6b).

5.3. Converting images

Figure 7 shows the conversion task of Section 3.3, performed with the new interface. The first two iterations have been demonstrated as they were before, including several noise events (Figure 7a,b); Familiar correctly predicts the next complete iteration.

6. Discussion

Familiar is the first PBD system that works with unmodified applications on a commercial platform. Other such systems have required completely new applications [1], modifications to the applications and environment [2], are application-specific [3,4], operate at a low level [4], or are macro recorders that do not perform inferencing. Familiar achieves generality by leveraging existing scripting systems, languages, and learning techniques.

Testing with end users revealed a variety of problems with the interface. Solutions to almost all of these have been incorporated into the most recent version, Future work will focus on termination conditions, and evaluating the new interface.

The evaluation exercise described in this paper shows that end users can use PBD, but highlighted a range of difficulties that led to improvements in the interface. Our findings should prove instructive to those who create the next generation of interface agents.

References

[1] Cypher A. (Ed) *Watch what I do: Programming by demonstration*. MIT Press, 1993.

[2] Cypher A. "Eager: Programming Repetitive Tasks by Demonstration." In [2], pp 205–217, 1993.

[3] Lieberman, H. "Integrating User Interface Agents with Conventional Applications", *Proc. Intelligent User Interfaces*, ACM Press, 1998.

[4] Masui T. and Nakayama K. "Repeat and Predict — Two Keys to Efficient Text Editing", *Proc. CHI*, 1994.

[5] Paynter, G. W. *Automating Iterative Tasks with Programming by Demonstration*. PhD Thesis, CS Department, University of Waikato, NZ, 2000.