Working Paper Series
ISSN 1170-487X

Automating iterative tasks with programming by demonstration: a user evaluation

by **Gordon W Paynter and Ian H Witten**

Working Paper 99/7
May 1999

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

Automating iterative tasks with programming by demonstration: a user evaluation

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

ABSTRACT
Computer users often face iterative tasks that cannot be automated using the tools and aggregation techniques provided by their application program: they end up performing the iteration by hand, repeating user interface actions over and over again. We have implemented an agent, called Familiar, that can be taught to perform iterative tasks using programming by demonstration (PBD). Unlike other PBD systems, it is domain independent and works with unmodified, widely-used, applications in a popular operating system. In a formal evaluation, we found that users quickly learned to use the agent to automate iterative tasks. Generally, the participants preferred to use multiple selection where possible, but could and did use PBD in situations involving iteration over many commands, or when other techniques were unavailable.

KEYWORDS
programming by demonstration, aggregation, iterative tasks, user evaluation.

1. INTRODUCTION
Computers excel at performing monotonous tasks again and again, and can spare us hours of drudgery. But it is not always obvious just how to make a particular computer repeat a particular task. Many applications provide aggregation techniques and specialised tools for this purpose, but even when they are available these are not powerful enough for many tasks. The general solution of writing a program is beyond the skills, or interests, of most end-users. Ultimately, they find themselves repeating the same actions over and over again.

An experienced programmer might write a program or script to automate iterative tasks, but this solution is unavailable to inexperienced users and non-programmers. Programming by demonstration (Cypher 1993) is an end–user programming technique that can be used to solve iteration problems. PBD lets the user "program" the computer by showing it examples of what they want done, much as they might teach another person. Unlike conventional programming, users do not have to describe the entire task in advance, in flawless detail, in an abstract form, in a foreign environment. A good example is Eager (Cypher 1991), a PBD agent that watches what the user does, detects repetition, predicts future commands, suggests actions, and, ideally, completes the task on the user's behalf.

Complex demonstrational techniques have rarely been tested in real-life situations; those that have tend to be specialised for tasks like interface building (Myers 1993) or teaching (Lieberman 1993). Typical end-user applications have been either been specially implemented (Halbert 1993, Kurlander 1993, Modugno and Myers 1997) or extended in research environments (Cypher 1991, Myers 1998). Efforts to add PBD to existing applications (Kosbie and Zeiliger 1997, Lieberman 1998) are comparatively recent, but important for evaluating demonstrational techniques with real users.

Our work is motivated by the questions these interfaces have left unanswered. Can PBD be added to existing 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 use it? Will they prefer it to alternative techniques? To answer these, we have designed and implemented Familiar, a general-purpose, application-independent PBD system on a popular computer platform, and evaluated it by asking users to automate iterative tasks.

This paper addresses the problem posed by iterative tasks in common applications, and compares solutions based on PBD and multiple selection. The next section describes the problem domain and potential solutions. An overview

of Familiar's interface and use is given in Section three. Section four describes an evaluation of the system and records the results. The paper concludes with a discussion of our observations and of related tools and research.

2. ITERATIVE TASKS

Iterative tasks are those where users repeat a set of actions without interleaving actions from other tasks. For example, a user who works through a series of records in a database, entering a value in a particular field of each and immediately moving to the next, is performing an iterative task.

2.1 Pilot study: end-users and iteration

We conducted a brief pilot study to simulate, in a realistic applications context, situations where users are unable to find or deploy suitable tools for automating iteration. We asked six people to complete two or three iterative tasks (such as reformatting a text document and creating a spreadsheet) in applications they had used before. They were first asked to perform the tasks as they would in everyday computer use, and most did so with ease. They were then instructed to repeat the tasks with restrictions that prevented them from using first specialised tools and then multiple selction, forcing them to confront the repetitive nature of the tasks.

We observed, without surprise, that users do not like performing iterative tasks. Most were familiar with several tools for automating or avoiding iteration. When they were unable to use known techniques, they actively searched for new automation tools and strategies. When they found an acceptable strategy they tended to reuse it, even on occasions when they suspected that more efficient strategies might exist. This behaviour illustrates the users' understanding of the trade-off between the cost of time spent finding and learning a new technique (and the risk of failure), and the benefit of time saved by the new technique. When participants were finally forced to perform the repetition manually, they became bored and made mistakes.

2.2 Solving iterative tasks

In a study of CAD users, Bhavnani and John (1998) observed that iterative tasks can be divided into those that require iteration over commands, over data elements, or over both. Efficiencies are gained by using strategies and tools to aggregate elements and commands, so that one command can be applied to many elements, or many commands can be applied at once.

Multiple selection is a technique for aggregating objects. The user selects a group of objects and applies a single command to them all: the effect is the same as applying the command to each object individually.

Macro recorders, a simple example of PBD, let the user aggregate commands by "recording" a series of keystrokes and mouse movements and "playing" them when requested. When the user records a macro, they are demonstrating a simple program that will be executed when the macro is invoked. Macros are of limited use because each playback is identical; there can be no variation between repetitions.

Bhavnani and John found that even experienced professionals working in well-understood applications use sub-optimal strategies for solving iterative tasks. Many such tasks are encountered in areas where the user is unfamiliar with the commands available, or by inexperienced users. In these circumstances it seems unlikely indeed that the user will be able to identify the best strategy or locate the particular tool for automating a task. Further, it is impossible to automate repetition in some situations because there is no appropriate tool, or the tool cannot be applied in a suitable manner.

PBD can help in any of these situations. If the user knows how to perform a task—any task—they can also demonstrate that task, and a suitable PBD system can attempt to automate it even if the user has no conventional programming skills. The task need not be demonstrated in the best or most efficient way. It is of little importance to the user whether the demonstrated program uses two actions or ten to complete a task if it only takes one action to invoke it. PBD offers efficiency gains to users not by minimising the number of actions required, but by letting them delegate those actions to the computer, which it then performs autonomously.

Like macro recorders, PBD systems generally aggregate commands. They learn a series of actions that can then be applied to any object. Some also aggregate elements by learning which data they should be applied to.

3. THE "FAMILIAR" PBD SYSTEM

Familiar is an agent for automating iterative tasks in common applications. It uses a high-level event trace, the data the user is working on, and domain knowledge reported by the application to predict the actions the user will perform next.

3.1 The Familiar tutorial

Users are trained to use Familiar by working through a tutorial where they are asked to perform a simple iterative task: given a folder of 26 files, move them into a row across the screen.

The first step is to open the *Familiar* menu, which appears in the menubar of every application next to the *Help* menu, and choose *Start recording*. The user is then asked to go to the "tutorial example" folder in the Finder (the Macintosh operating system) and put file "a" in the top left hand corner of the window, as shown in Figure 1a.

When the user finishes the command a new window, labelled *Familiar history*, is created. This displays every AppleScript command that Familiar has recorded. At this point in the tutorial it lists three (Figure 1b), corresponding to the high-level events the user has performed: activating the Finder, selecting file "a," and

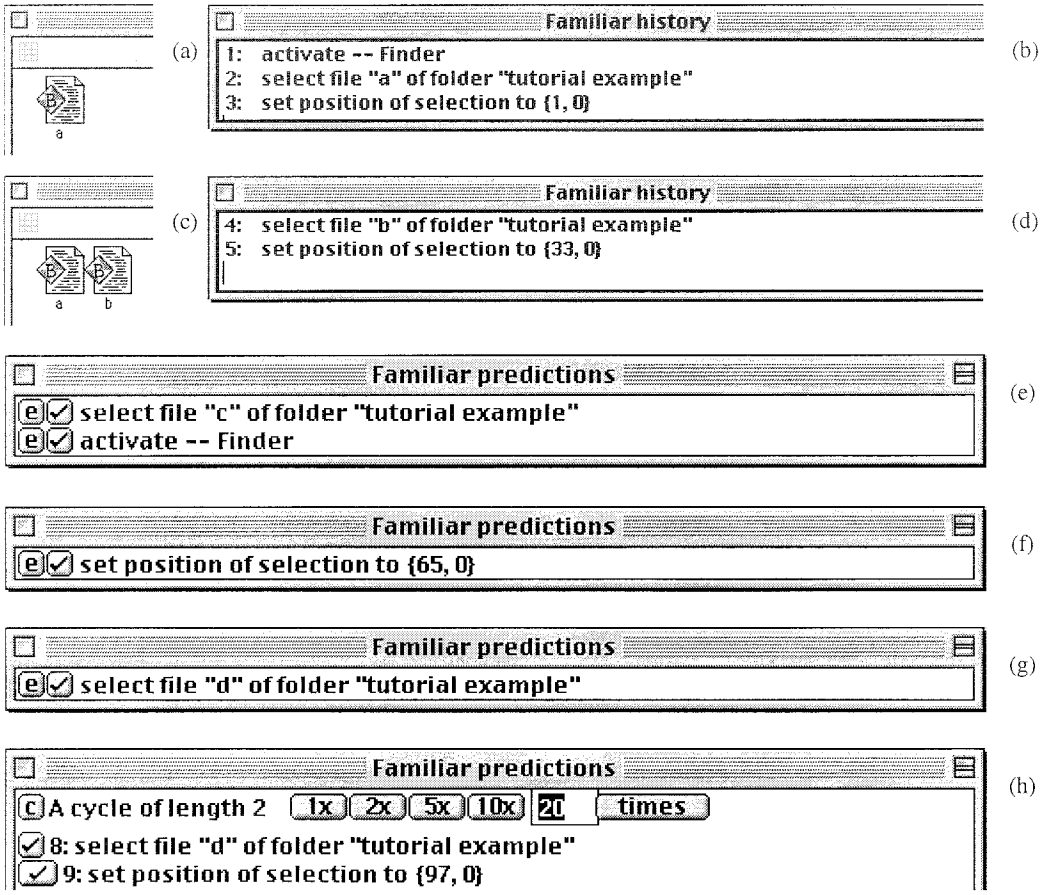

Figure 1: Using Familiar to automate a simple task. The user demonstrates the first iteration (a,b) and the second (c,d), and Familiar predicts the next three steps (e,f,g). The user expands the final prediction (g) to display the entire cycle (h).

setting the position of that selection. The user then places file "b" next to file "a" (Figure 1c). Again, their commands are added to the *history* window (Figure 1d).

At this point in the demonstration, Familiar has recorded two iterations of a cycle and therefore begins predicting. A new window titled *Familiar predictions* is created (Figure 1e) which displays two predictions. The first of these is correct—the next action the user intends to take is to select file "c". The second prediction is made by assuming that the *activate* command is an important step in the cycle, but that the user, through oversight, forgot to include it in the second demonstration.

There are two ways to ask Familiar to execute a prediction: click on the text of the command or on the *tick* button next to it. When the user clicks the first command in Figure 1e, several 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. The user can see file "c" being selected in the background. Third, the command is recorded by Familiar and added to the *history* window, 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 file (Figure 1f), and the user selects it. After the selection is repositioned, a new prediction is made, that the user will select file "d" (Figure 1g).

The user does want to select file "d", and it would be easy to tell Familiar to do so, but asking the agent to execute one step at a time is inefficient—it is probably faster for the user to move the files by hand. Next to the *tick* button in the *prediction* window is the *expand* button (labelled "e"). When it is clicked, Familiar displays the repeated cycles it detected that start with that command (Figure 1h).

In this case the topmost cycle consists of a *select* command and a *set* command. The 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). The user has two ways to make Familiar perform a complete cycle of the task. One is to click on the last command in the cycle (in this case, step 9). Clicking on any command in a cycle will execute each of the commands up to and including that one. The other is to click on the *one times* button (labelled "1x").

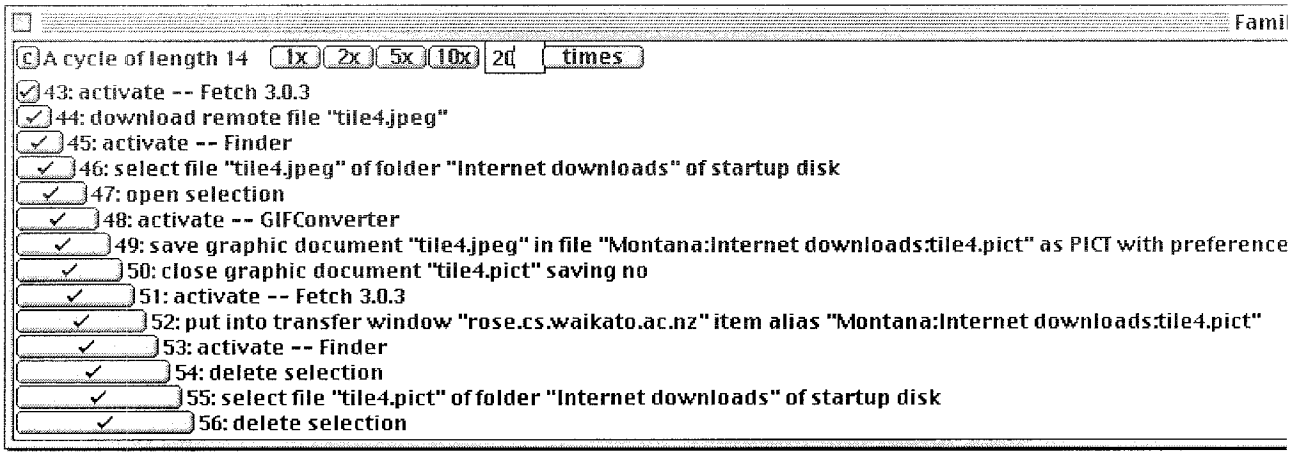

Figure 2: Using Familiar to automate the Image task.

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 displays its predictions for the next iteration of the cycle.

Now that the user has taught Familiar the task, the remainder of the tutorial asks the user to experiment with the controls in the prediction window. To make Familiar perform more than one command at a time, the user can either press the *two times* ("2x"), *five times* ("5x"), or *ten times* ("10x") buttons, or type an arbitrary number of repetitions in the field provided and press the button marked *times*. This field is also used to display the number of cycles that Familiar has still to complete when the user has asked for more than one iteration.

When the user has finished the task, they choose *End task* from the *Familiar* menu.

3.2 Implementation

Familiar was written as an experimental platform to test a number of hypotheses. One aspect we were interested in was the ability to add PBD to existing applications so that we could examine the users behaviour in these conditions.

Platform

Familiar is written for the Apple Macintosh in Macintosh Common Lisp (version 4.2) and communicates using the AppleScript scripting language. The evaluation described in this paper uses AppleScript (version 1.1.2) with the MacOS Finder (version 8.1), Microsoft Excel (version 5.0), Fetch (version 3.0.3) and GIFConverter (version 2.4d18) on a 200Mhz Power Macintosh computer (model 7300/200).

AppleScript is a suitable platform for this research because it allows us to examine and control other applications, and to record the users actions as a series of high-level events as they are performed. Another important feature is the English–like command syntax; as it is designed to be readable by typical users, it can be used to describe commands and objects to the user.

AppleScript has several weaknesses. The first is speed—the version of AppleScript used in the evaluation is slow. Making an application scriptable is an extra cost to the application developer, so not all applications are scriptable and fewer still are recordable. AppleScript is added independently to each application by different programmers, so the quality of the implementations vary; some are poor. AppleScript monopolises the interface and the user often cannot control the machine while it is in use. Finally, AppleScript is intended for human programmers and lacks some abilities that would make PBD much simpler, particularly commands for exploring application data structures at run-time. These problems are canvassed in more detail in Paynter 1999.

Inferencing

Familiar's inferencing is domain-independent, and based on detecting cycles of commands in the event trace and extrapolating those cycles. A full description is available in Paynter and Witten 1999.

4. EVALUATION

The aim of the evaluation is to gauge the extent to which actual users will choose to invoke the PBD interface to automate iterative tasks, and to compare it with two other means of solving repetitive tasks: using multiple selection, and by hand.

A group of users was asked to perform four variations of two iterative tasks, and we observed the strategies they used to solve them. Ten university students or recent graduates participated in the evaluation. Three were from computer-related disciplines, and three had subsequently used computers professionally.

Each participant was first asked to perform the task the way they would if doing it for themselves (variant 1). The participants were asked to repeat the task with the restriction that they may not use multiple selection (variant 2). They were then shown Familiar, and asked to

Variant	Number of users	Used multiple selection	Used Familiar	Average complete	Average actual time	Average estimated time
1	10	9	0	93%	5:01	6:03
2	9	0	0	59%	6:28	6:43
3	9	0	9	40%	16:10	10:09
4	4 (8)	8	3	83%	6:59	11:20

Table 1: Selected results for the Calendar task

work through the tutorial described above. Finally, they were asked to repeat the task twice more, once without multiple selection but with Familiar available (variant 3), and once with the choice of any technique (variant 4).

4.1 The iterative tasks

In the *Calendar task*, the participant is asked to duplicate a printed calendar as a Microsoft Excel spreadsheet. The calendar covers a six-month period and consists of a row of month headings at the top, a column of day headings on the left, and a body of dates. (In total, 225 cells are used.) The calendar can quickly be built with multiple selection and the spreadsheet's Fill tools, but is tedious to enter by hand. In the third and fourth variations the calendar is altered so that the dates are in the leftmost columns and the days make up the body of the spreadsheet.

The *Image task* is more complex. It uses three different programs, and only one of the participants had used all of them. In the first two variations, the participant is asked to use a graphical FTP client to download twelve files from an FTP site onto the hard drive; to use an image manipulation program to resize them; to use the FTP client to return the modified files to the FTP site; and to use the operating system to clean up any copies left on the local machine. In the last two variations the participants were asked to process 62 files instead of 12, and instead of resizing each they were asked to convert them from one graphics format to another.

The Calendar task involves cycles of only two high-level events, repeated on 225 (or 221) cells. The Image task requires approximately 15 high-level events for each of 12 (or 62) images. Figure 2 shows a cycle of 14 events that the user has created to automate the Image task.

4.2 Calendar task results

Table 1 shows how many participants attempted the four variations of the Calendar task, and summarises the techniques they used. One participant did not use multiple selection (she had little experience with Excel), disqualifying her from variations two and four, and another was unable to participate in the last two variations due to technical difficulties. Only four subjects attempted task four due to time constraints, but the remaining four were asked to explain how they would finish the task.

Every participant used multiple selection whenever possible, except for the one who never used it. Some used multiple selection with the *Copy* and *Paste* commands, some with the *Fill* commands, and some used combinations of these methods (for example, using Fill to generate a column of dates, then copying it into another column). Only one participant used formulas, and then only in one variant.

Each participant used Familiar in preference to typing the Calendar by hand, but only three used it when they had the option of using multiple selection as well. Of these, two used Familiar to generate a column of day names, then used multiple selection to copy the selection to other columns; and the other attempted to use Familiar to copy and paste seven selected cells (one week). He quickly found that Familiar was not suited to this task, and stopped using it. (A fourth participant, who did not complete the task due to time constraints, said he might use Familiar for parts of it, but could not explain how).

The Calendar task is time-consuming, particularly the variations that restrict aggregation. Participants were often stopped partway through when it became apparent how they were going to finish the task. It is interesting to consider the average percentage of the task—estimated by the number of cells filled—the participants finished and the time it took them to do so. (These approximate completion rates are not directly comparable because some cells can be filled faster than others, the first variants are different tasks to the later ones, and many participants spent a long time analysing the problem before they began filling cells.)

Table 1 shows that participants, on average, took longer and completed less of the task when they used Familiar. The opposite is true of multiple selection. The principal reason for this result, identified by all the participants, is Familiar's speed. It is very slow; a problem caused by our use of AppleScript to communicate with other applications.

Although they tended to realise that the third variant was taking longer than the others, users underestimated the amount of time it did in fact take. In spite of this, the participants still opted to use Familiar. When asked, some thought Familiar was taking longer than it would to perform the task by hand, but they still preferred to use it

Variant	Number of users	Used Multiple Selection	Used Familiar	Taught Familiar task	Average complete	Average actual time	Average estimated time
1	10	9	0	0	94%	7:34	8:00
2	6	0	0	0	83%	5:16	6:10
3	10	0	10	9	25%	19:10	15:11
4	7 (9)	6	9	6	15%	8:18	8:40

Table 2: Selected results for the Image task

because they were spared having to physically make the repetitive actions. Some participants were asked if they would type all the cell values in variant 3 if they could be sure it was faster. Their replies included "No... it's more convenient not to have to type it... typing it gives you the chance to make mistakes, too" and "I'd probably still use Familiar, but it depends if I were in a rush to finish or not."

4.3 Image task results

Table 2 summarises the participant's performance on the Image task. One did not use multiple selection in variant 1, and three used very little; these participants were not asked to perform variant 2. All the participants attempted variant 3, but only seven attempted variant 4; of the other three, one used no multiple selection in the first variant, and the other two had run out of time. These two described how they planned to perform the task.

The first variation of the task generally took longer than the second despite the fact that almost all the participants used multiple selection when it was available. There were two reasons for this anomaly. One step in the cycle (*resize* the image in the first two variants, and *Save As* in the latter two) cannot be automated with multiple selection, and must be performed separately for each image. There were relatively few data elements and a pronounced learning effect—by the second variant most participants had adopted keyboard shortcuts where they were available.

In the last two variations of the task 62 images were used instead of 12. Again the participants took longer when using Familiar, and tended to underestimate the time they spent. In this case, however, the completion figures are particularly misleading. In both variants, all but one participant used Familiar and could have finished the task by entering the appropriate number of cycles and telling Familiar to execute them. They were prevented from so doing, because the AppleScript commands would take too long to execute.

The one participant who did not complete the last two variants using Familiar presents an interesting case study. He used Familiar to automate the task in stages, first to download all the files, then to convert them all, and so on. He was successful for most of the task, but in both variants accepted an incorrect prediction while uploading the files that prevented him from completing the task correctly. As the participant was in a hurry, he did not check his results. The mistake was compounded by three factors: the participant was an inexperienced computer user; he poorly understood the AppleScript feedback; and he naively trusted Familiar's predictions (he remains unaware of his error).

When the participants had the option of using any technique, all nine elected to use Familiar, and six chose to use multiple selection as well. The portion of task performed with Familiar varied. One participant used multiple selection whenever it was possible and Familiar only to automate the *Save As* commands, while the three who ignored multiple selection taught Familiar a single iterative task comprised of 14 or 15 steps in three applications. An example of such a cycle is shown in Figure 2. Most participants took a middle course, using Familiar in some parts of the task and multiple selection in others.

5. DISCUSSION

5.1 Tool choice

The users were all able to apply Familiar in a range of domains and situations. All the participants used Familiar in preference to performing the tasks by hand, sometimes even when they felt they could have performed the task more quickly manually. Their stated reasons for this choice included ease of use (or laziness, as one participant put it), reduced potential for error, minimising physical actions (one participant suffered from repetitive strain injury), and Familiar's ability to work unsupervised.

All but one participant used multiple selection rather than perform the iterative tasks by hand. They were able to apply the technique in many domains and situations. The remaining participant was one of the least experienced, and we are confident that she too would have used multiple selection if she had had a brief introduction to it, as she did for Familiar.

When the participants were able to choose between the two tools, they generally preferred multiple selection. It was used more often than Familiar, and was used more quickly and with less planning by the participant. However, all the users found situations where PBD was

preferable. These included cases where multiple selection could not be applied to the objects of interest, tasks with longer cycles, and when it was not clear how (or if) multiple selection can be applied.

Some participants made heavier use of PBD than others. Generally they had no difficulty mixing the techniques, using each to perform those parts of the task they thought were appropriate.

5.2 User comments on Familiar

The most common complaint about Familiar was that it is slow. Familiar's speed is restricted by AppleScript, which is not a fast language. Familiar has been trialed on a faster machine using the latest release of the Finder (version 8.6) which includes a AppleScript (version 1.3.7) and the speed increase is significant enough to resolve many of these complaints.[1]

Users noted several other faults with Familiar. It occasionally made poor predictions in the face of multiple counter–examples or took too long to make predictions. The interface does not explain how predictions are made, and sometimes makes predictions based on generalisations that do not match the user's mental model, causing some confusion one occasion when the inconsistency was revealed.

Many participants commented that they thought Familiar was unsuitable for the short cycles in the Calendar task because it is too slow to execute predictions and learn changes in patterns. More experienced users said that even if Familiar were faster, multiple selection and the Autofill functions were superior tools because the user knew exactly what data would be entered (more specifically, they can clearly see the data from which predictions will be extrapolated) and because the termination conditions are set explicitly and visually.

Multiple selection, is more attractive than Familiar in many situations because it is fast, simple, polished, familiar to users, and able to set explicit termination conditions. A better implementation of PBD might neutralise the first three of these advantages, allowing a more balanced comparison.

[1]The version of AppleScript used in the evaluation (1.1.2) was written for Motorola 68000 microprocessors and run in a 68000 emulator by the operating system; while the new version (1.3.7) is written for the Power PC microprocessor used in Power Macintosh computers (ie: it is "native" code). It is unclear how much of the speed increase is due to the new machines faster architecture, how much is due to its faster clock speed (an increase from 200 to 400 MHz), and how much is due to the new AppleScript implementation.

5.3 Macro recorders

Macro recorders are a closely related technology to Familiar, so we were interested in examining the user's experience with these tools. We did so primarily through post-experiment interviews with each of the ten participants.

One of the participants inquired about using a Macro recorder during the experiment. This participant had used specific Macros on a daily basis in the workplace, and wished to aggregate the *resize*, *save*, and *close* steps of variation two of the Image tasks, but was told there was no Macro recorder available.

The interviews revealed that five of the ten participants had used a Macro recorder in the past. These participants were asked if they had considered using macros in the Image task, and all said but the one described above said no. Two had "forgotten" about Macro recorders, the other two said they didn't realise it was possible to use one. None of the participants thought a Macro was appropriate for the Calendar task because it was too "simple". When asked why thy didn't use macros, one participant said "I felt the time I would have taken to refresh myself with the Macro would have been longer than just doing the table." Another considered writing (programming) a Macro, but not recording (demonstrating) one, and added that his employer would probably discourage him from learning (to program) Macros in the workplace.

The participant's lack of experience with Macro recorders is instructive. Only one of the participants seriously considered using a Macro; others thought them inappropriate or too complex, and half did not know of them at all. We conclude that if demonstrational techniques like Familiar are to ever reach widespread use, they must be easy to discover and access, and easy to learn and use.

5.4 Related research

Eager

Familiar's nearest relative is Eager, an agent that helps automate tasks in the HyperCard multimedia environment (Cypher 1991). As the user works through an iterative task, Eager detects repetition and starts predicting the user's next commands and graphically highlighting them on the screen. When the user has seen enough to be confident that Eager has learned correctly, they can request that it finish the task for them. Eager showed that PBD can be used to solve iterative tasks that are not amenable to solutions using other aggregation tools.

Eager is a research system that requires the HyperCard application and parts of the operating system to be rewritten to facilitate prediction and graphical feedback. In principle it is domain independent, but this specialised programming prevents its use in other applications. Eager is neither robust nor general: it does not tolerate mistakes in the user's demonstration, and it restricts the user to

those actions and inferences that the programmer has anticipated. Familiar addresses all these shortcomings (to varying degrees) by interacting solely with AppleScript and sacrificing Eager's innovative user interface.

Cypher reports that users evaluating Eager were "uncomfortable giving up control" to the agent (Cypher 1991). The participants in our evaluation did not appear discomforted when asking Familiar to execute predictions. We speculate that this is due to differences in the design of the Familiar experiment: the participants were working in familiar environments; the use of the agent was entirely optional and the interface less intrusive than Eager's; and the tasks were large enough that the participants were motivated to seek some way to automate them. It is crucial that the user does not feel that the system has taken over, but that they have voluntarily delegated a menial task—one that they do not want to perform themselves—to Familiar.

Domain independence

Many PBD systems are application-specific. They are built into a particular application and cannot be used in another. Often this is intentional, as systems with greater knowledge of the domain of application can better help with specialised tasks. Unfortunately, it means that tasks that span applications cannot be automated, a new PBD system must be implemented in each program, and that researchers face the overhead of implementing entire new applications.

As a result, most PBD systems strive for domain-independence (or application-independence). Some are domain-independent in theory, but have not been implemented in more than one application (Eager, CIMA, AgentScript, Epiphyte). Others attempt to provide PBD in a range of applications.

One way to do so is to create a specialised programming environment, and let developers write programs that interact with a system-wide PBD engine. An example of this approach is AIDE, the Application-Independent Demonstrational Environment (Piernot & Yvon, 1993). Apple Macintosh SmallTalk developers can make their applications compatible with the PBD system by implementing a set of methods in their application. The advantages of this architecture are that developers can add PBD to their applications without creating an entire PBD system, and that the PBD system can exploit domain knowledge yet remain domain-independent. However, it does require that developers create or change their applications to be AIDE-aware, and work on a specific platform. Few are likely to go to this trouble for an untested technology like PBD.

Myers (1998) takes a similar approach with the Topaz PBD system. Topaz adds "scripting by demonstration" facilities to any graphical application developed with the Amulet development environment and command object architecture. Unlike AIDE, application developers do not have to explicitly make the program PBD-aware, they simply have to use the toolkit. Topaz does little inferencing and generalises by providing defaults, exploring the interface data structure, and soliciting the user's advice.

Marionette Strings

Lieberman (1998) describes programs like Familiar that allow an agent to manipulate unmodified applications by emulating the user in a "marionette strings" approach. His ScriptAgent is a PBD system that uses AppleScript recording to record and perform commands. The user demonstrates transformations to an input object, and ScriptAgent builds a program to transform any object in the same way. It works in the Macintosh Finder, but the technique could, in principle, be extended to other AppleScript recordable applications, though it is not clear how the agent interface would be integrated with other off-the-shelf applications. Lieberman concludes that Marionette string approaches like ScriptAgent and Familiar can work, but would be much improved by building applications that support better integration of application and agent data.

Zeiliger and Kosbie (1998) have described a similar approach for automating frequent tasks in unmodified applications. They monitor the user by trapping low-level and mid-level events, which are aggregated into high level events using an external model of the application. Unlike Familiar, their system is only partially automated, and a human expert user plays an important role in analysing the event histories and maintaining the application model. The user interface for automating tasks is described, so it difficult to discuss this system in detail.

7. CONCLUSIONS

We have compared techniques for automating iterative tasks using PBD and multiple selection, examples of command and element aggregation. We found that users can and will use PBD when the alternative is no aggregation at all, but that multiple selection is often preferable to our solution because it is simple, fast, and sets termination conditions cleanly. The PBD solution is more useful in tasks that involve many commands in each cycle, and in situations where the users cannot apply multiple selection, or has difficulty doing so.

Familiar is an agent for automating repetitive tasks that works with a range of applications in the Macintosh computer. The agent exploits existing technology to examine, monitor, access, and control applications, and does not require any modification or description of the applications or operating system. It will adapt to suitable new applications as they are encountered. The agent has several shortcomings, but is general, domain-independent, robust, and simple to both learn and use.

REFERENCES

Bhavnani, S. K. and John, B. E. (1998). Delegation and Circumvention: Two Faces of Efficiency. In *Proceedings of CHI '98.*

Cypher A. (1991) Eager: Programming Repetitive Tasks by Example. In *Proceedings of CHI '91.* (A modified version appears in Cypher 1993.)

Cypher A. (Ed) (1993) *Watch what I do: Programming by Demonstration.* MIT Press.

Halbert D. (1993) SmallStar: Programming by Demonstration in the Desktop Metaphor. In *Cypher 1993*, pp 103–124.

Kosbie D. and Zeiliger R. (1997) Automating Tasks for Groups of Users. In *Proceedings of INTERACT'97*, Sydney, Australia.

Kurlander, D. (1993) Chimera: Example-based graphical editing. In *Cypher 1993*, pp 271–292.

Lieberman, H. (1993) Tinker: A Programming by Demonstration System for Beginning Programmers. In *Cypher 1993*, pp 49–68.

Lieberman, H. (1998) Integrating User Interface Agents with Conventional Applications. In *Proceedings of IUI '99.*

Modugno F. and Myers B. A. (1997) Visual Programming in a Visual Shell—A Unified Approach. *Journal of Visual Languages and Computing, 8, 5/6*, pp491522.

Myers B. A. (1993) Garnet: Uses of Demonstrational Techniques. In *Cypher 1993*, pp 219–238.

Myers B. A. (1998) Scripting Graphical Applications by Demonstration. In *Proceedings of CHI '98.*

Paynter G. W. (1999) Familiar: automating repetition in common applications. In *Proceedings of the 3rd New Zealand Computer Science Research Students Conference.* The University of Waikato, New Zealand.

Paynter G. W. and Witten I. H. (1999) Automating Iteration with Programming by Demonstration: Learning the User's Task. In *Proceedings of the Workshop on Learning About Users, IJCAI '99.*

Piernot P. P., and Yvon M. P. (1993) The AIDE project: an application-independent demonstrational environment. In *Cypher 1993*, pp 383–401.