Color profile: Generic CMYK printer profile
Composite Default screen

Chapter Nineteen

Visual Generalization in Programming by Example

ROBERT ST. AMANT AND LUKE ZETTLEMOYER
North Carolina State University

HENRY LIEBERMAN,
Massachusetts Institute of Technology

RICHARD POTTER,
Japan Science and Technology Corporation

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:37 PM
TNT Job Number: [**002564**] • *Author:* [**Lieberman**] • *Page:* **371**

Color profile: Generic CMYK printer profile
Composite Default screen

Abstract

In programming by example (PBE; also sometimes called programming by demonstration) systems, the system records actions performed by a user in the interface and produces a generalized program that can be used later in analogous examples. A key issue is how to describe the actions and objects selected by the user, which determines what kind of generalizations will be possible. When the user selects a graphical object on the screen, most PBE systems describe the object using properties of the underlying application data. For example, if the user selects a link on a Web page, the PBE system might represent the selection based on the link's HTML properties.

In this chapter, we explore a different, and radical, approach—using visual properties of the interaction elements themselves, such as size, shape, color, and appearance of graphical objects—to describe user intentions. Only recently has the speed of image processing made feasible real-time analysis of screen images by a PBE system. We have not yet fully realized the goal of a complete PBE system using visual generalization, but we feel the approach is important enough to warrant presenting the idea.

Visual information can supplement information available from other sources and opens up the possibility of new kinds of generalizations not possible from the application data alone. In addition, these generalizations can map more closely to the intentions of users, especially beginning users, who rely on the same visual information when making selections. Finally, visual generalization can sometimes remove one of the worst stumbling blocks preventing the use of PBE with commercial applications—that is, reliance on application program interfaces (APIs.). When necessary, PBE systems can work exclusively from the visual appearance of applications and do not need explicit cooperation from the API.

19.2 If You Can See It, You Should Be Able to Program It

Every PBE system has what Halbert (1993) calls the "data description problem": when users select an object on the screen, what do they mean by it? Depending on how you describe an object, it could result in very different effects the next time you run the procedure recorded and generalized by the system. During a demonstration to a PBE system, if you select an icon for a file foo.bar in a desktop file system, did you mean (1) just that specific file

S
R
L

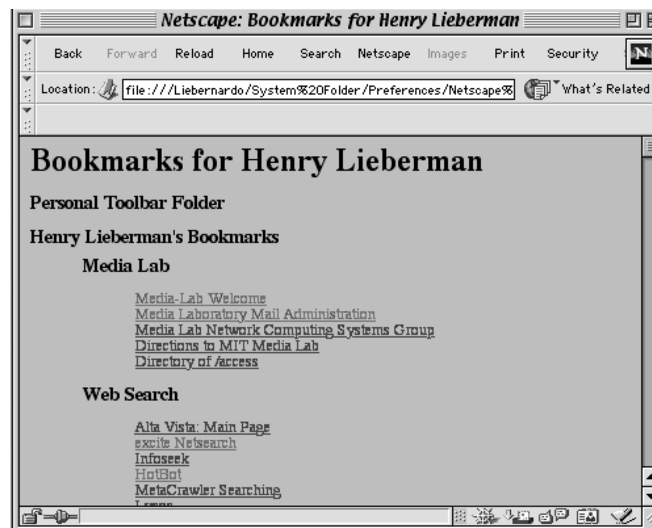
Color profile: Generic CMYK printer profile
Composite Default screen

and no other? (2) Any file whose name is foo.bar? (3) Any icon that happened to be found at the location where you clicked?

Most systems deal with this issue by mapping the selection onto the application's data model (a set of files, email messages, circles and boxes in a drawing, etc.). They then permit generalizations on the properties of that data (file names, message senders, etc.). But sometimes the user's intuitive description of an object might depend on the actual visual properties of the screen elements themselves—regardless of whether these properties are explicitly represented in the application's command set. Our proposal is to use these visual properties to permit PBE systems to do "visual generalization."

For an example of why visual generalization might prove useful, suppose we want to write a program to save all the links on a Web page that have not been followed by the user at a certain point in time (Fig. 19.1). If the Netscape browser happened to have an operation "Move to Next Unfollowed Link" available as a menu option or in its API, we might be able to automate the activity using a macro recorder such as Quickeys. But, unfortunately, Netscape does not have this operation (nor does it even have a Move to the Next Link operation). Even if we had access to the HTML source

FIGURE **19.1**



Can we write a program to save all the unfollowed links?

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:38 PM
TNT Job Number: [**002564**] • *Author:* [**Lieberman**] • *Page:* **373**

Color profile: Generic CMYK printer profile
Composite Default screen

of the page, we still wouldn't know which links had been followed by the user. This is a general problem for PBE systems in interfacing to almost all applications. Interactive applications make it easy for users to carry out procedures and do not expect to be treated as a subroutine by an external system.

This example shows the conceptual gap between a user's view of an application and its underlying programmable functionality. Bridging this gap can be extremely difficulty for a PBE system—its representation of user actions may be a complete mismatch for the user's actual intentions. But perhaps we are looking at this problem from the wrong perspective. From the user's point of view, the functionality of an interactive application is defined by its user interface. The interface has been carefully developed to cover specific tasks; to communicate through appropriate abstractions; and to accommodate the user's cognitive, perceptual, and physical abilities. A PBE system might gain significant benefits if it could work in the same medium as a user, if it could process the visual environment with all its information. This is the key insight we explore in this chapter.

19.3 What Does Visual Generalization Buy Us?

Let's imagine a PBE system that incorporates techniques to process a visual interactive environment, to extract information potentially relevant to the user's intentions. What does the system gain from these capabilities?

- *Integration into existing environments:* Historically, most PBE systems have been built on top of isolated research systems, rather than commercial applications. Some have been promising but have not been adopted because of the difficulty of integration. A visual PBE system, independent of source code and API constraints, could potentially reach an unlimited audience.
- *Consistency:* Independence of an application's source code or API also gives a PBE system flexibility. Similar applications often have similar appearance and behavior; for example, users switch between Web browsers with little difficulty. A visual PBE system could take advantage of functional and visual consistency to operate across similar applications with little or no modification.
- *New sources of information:* Most important, some kinds of visual information may be difficult or impossible to obtain through other means.

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:38 PM
TNT Job Number: [002564] • Author: [Lieberman] • Page: 374

Color profile: Generic CMYK printer profile
Composite Default screen

Furthermore, this information is generally closely related to the user's understanding of an application.

These are all benefits to the developers of a PBE system, but they apply equally well to the users of a PBE system. In the Netscape example, a visual PBE system would be able to run on top of the existing browser, without requiring the use of a substitute research system. Because Netscape has the convention of displaying the followed links in red and the unfollowed links in blue, a user might specify the "Save the Next Unfollowed Link" action in visual terms as "Move to the next line of blue text, then invoke the Save Link As operation." This specification exploits a new, visual source of information. Finally, the general consistency between browsers should allow the same system to work with both Netscape and Microsoft Internet Explorer, a much trickier proposition for API-based systems.

Providing a visual processing capability raises some novel challenges for a PBE system:

- *Image processing:* How can a system extract visual information at the image-processing level in practice? This processing must happen in an interactive system, interleaved with user actions and observation of the system, which raises significant efficiency issues. This an issue of the basic technical feasibility of a visual approach to PBE. Our experience with VisMap (described later) shows that real-time analysis of the screen is feasible on today's high-end machines.
- *Information management:* How can a system process low-level visual data to infer high-level information relevant to user intentions? For example, a visual object under the mouse pointer might be represented as a rectangle, a generic window region, or a window region specialized for some purpose, such as illustration. A text box with a number in it might be an element of a fill-in form, a table in a text document, or a cell in a spreadsheet. This concern is also important for generalization from low-level events to the abstractions they implement: is the user simply clicking on a rectangle or performing a confirmation action?
- *Brittleness:* How can a system deal gracefully with visual variations that are beyond the scope of a solution? In the Netscape example of collecting unfollowed links, users may, in fact, change the colors that Netscape uses to display followed versus unfollowed links, thereby perhaps obsoleting a previously recorded procedure. A link may in fact extend over more than a single line of text, so that the mapping between lines and links is not exact. Similar blue text might appear in a GIF image

S
R
L

Color profile: Generic CMYK printer profile
Composite Default screen

and be inadvertently captured by the procedure. And, if the program is visually parsing the screen, links that do not appear because they are below the current scrolling position will not be included. Out of sight, out of mind! The latter problem might be cured by programming a loop that scrolled through the page as the user would. Most of these problems are put in a novel light if we observe that they can be difficult even for a human to solve. Almost everyone has been fooled now and then by advertising graphics that camouflage themselves as legitimate interface objects; without further information (such as might be provided by an API call), a visual PBE system cannot hope to do better.

19.4 Low-Level Visual Generalization

Potter's work on pixel-based data access pioneered the approach of treating the screen image as the source for generating descriptions for generalization. The TRIGGERS system (Potter 1993) performs exact pattern matching on screen pixels to infer information that is otherwise unavailable to an external system. A "trigger" is a condition-action pair. For example, triggers are defined for such tasks as surrounding a text field with a rounded rectangle in a drawing program, shortening lines so that they intersect an arbitrary shape, and converting text to a bold typeface. The user defines a trigger by stepping through a sequence of actions in an application, adding annotations for the TRIGGERS system when appropriate. Once a set of triggers have been defined, the user can activate them, iteratively and exhaustively, to carry out their actions.

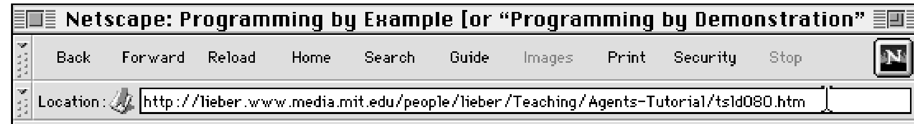
Several strategies can be used to process visual pixel information so that it can be used to generalize computer programs. The strategy used by TRIGGERS is to compute locations of exact patterns within the screen image. For example, suppose a user records a mouse macro that modifies a URL to display the next higher directory in a Web browser (Figure 19.2). Running the macro can automate this process, but only for the one specific URL because the mouse locations are recorded with fixed coordinates. However, this macro can be generalized by using pixel pattern matching on the screen image. The pattern to use is what a user would look for if doing the task manually: the pixel pattern of a slash character. Finding the second to the last occurrence of this pattern gives a location from which the macro can begin the macro's mouse drag, which generalizes the macro so that it will work with most URLs.

S
R
L

Color profile: Generic CMYK printer profile
Composite Default screen

FIGURE **19.2**

1. Select URL text field.

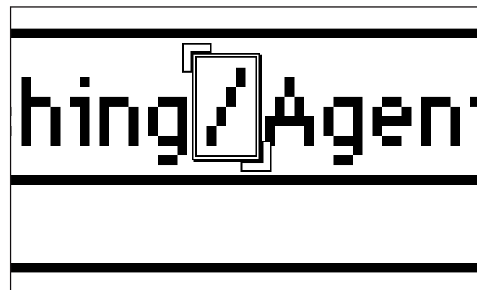


2. Start mouse drag.

3. Finish mouse drag.

4. Delete selection.

The generalizing pixel pattern:



Steps in a mouse macro to move a browser up one directory, and selecting a pixel pattern that can generalize the macro.

Even though this macro program affects data such as characters, strings, URLs, and Web pages, the program's internal data is only low-level pixel patterns and screen coordinates. It is the *use* within the rich GUI context that gives higher-level meaning to the low-level data. The fact that a low-level program can map so simply to a much higher-level meaning attests to how conveniently the visual information of a GUI is organized for productive work. Potter (1993) gives more examples.

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:44 PM
TNT Job Number: [**002564**] • *Author:* [**Lieberman**] • *Page:* **377**

Color profile: Generic CMYK printer profile
Composite Default screen

The advantage of this strategy is that the low-level data and operators of the programming system can map to many high-level meanings, even ones not originally envisioned by the programming system developer. The disadvantage is that high-level internal processing of the information is difficult, since the outside context is required for most interpretation.

Another system that performs data access at the pixel level is Yamamoto's (1998) AutoMouse, which can search the screen for rectangular pixel patterns and click anywhere within the pattern. Copies of the patterns can be arranged on a document and connected to form simple visual programs. Each pattern can have different mouse and keyboard actions associated with it.

19.5 High-Level Visual Generalization

Zettlemoyer and St. Amant's (1999) VisMap is in some ways a conceptual successor to TRIGGERS. VisMap is a programmable set of sensors, effectors, and skeleton controllers for visual interaction with off-the-shelf applications. Sensor modules take pixel-level input from the display, run the data through image-processing algorithms, and build a structured representation of visible interface objects. Effector modules generate mouse and keyboard gestures to manipulate these objects. VisMap is designed as a programmable user model, an artificial user with which developers can explore the characteristics of a user interface.

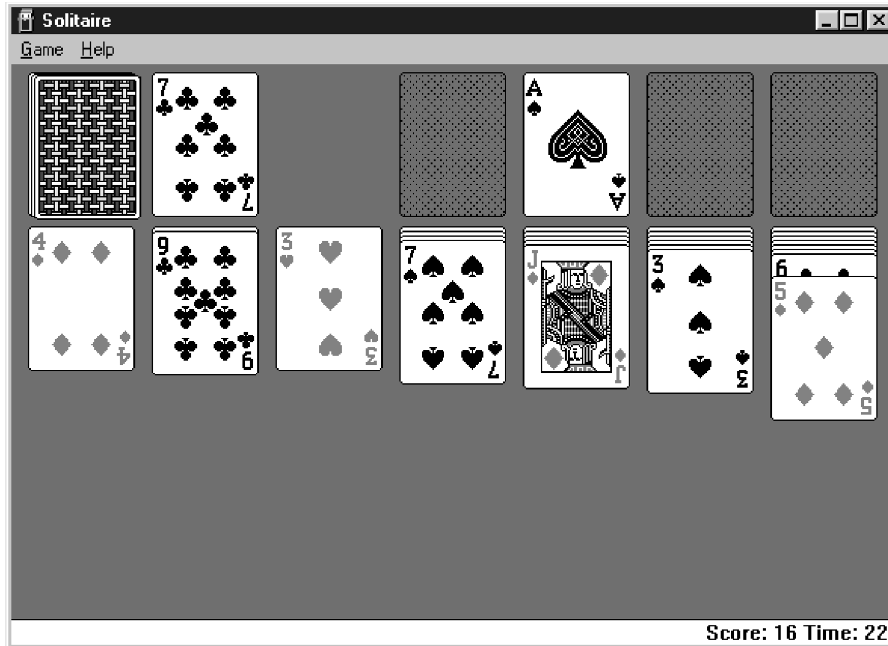
VisMap is not, by itself, a PBE system. But it does demonstrate that visual generalization is practical in an interface, and we hope to apply its approach in a full PBE system. VisMap translates the pixel information to data types that have more meaning outside the GUI context. For example, building on VisMap we have developed VisSolitaire, a simple visual application that plays Microsoft Windows Solitaire. VisMap translates the pixel information to data types that represent the state of a generic game of Solitaire. This state provides input to an AI planning system that plays a reasonable game of solitaire, from the starting deal to a win or loss. It does not use an API or otherwise have any cooperation from Microsoft Solitaire.

VisSolitaire's control cycle alternates between screen parsing and generalized action. VisSolitaire processes the screen image to identify cards and their positions. When the cards are located, a visual grammar characterizes them based on relative location and visual properties. In this way the system can identify the stacks of cards that form the stock, tableau, and

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:44 PM

Color profile: Generic CMYK printer profile
Composite Default screen

FIGURE 19.3



Visual Processing Results

Stock:	Tableau:	Foundation:
(:BACK)	(0 :BEHIND)(4 :DIAMONDS)	(:BLANK)
(7:CLUBS)	(1 :BEHIND)(9 :CLUBS)	(:ACE :SPADES)
	(0 :BEHIND)(3 :HEARTS)	(:BLANK)
	(4 :BEHIND)(:JACK :DIAMONDS)	(:BLANK)
	(5 :BEHIND)(3 :SPADES)	

VisSolitaire source data and visual processing results.

foundation, as well as classify each card based on visual identification of its suit and rank, as shown below in Figure 19.3.

A bottom-up pattern recognition process is interleaved with a top-down interpretation of the visual patterns. Key to the effectiveness of the system is the loose coupling between these two components. The strategic, game-

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:50 PM
TNT Job Number: [**002564**] • *Author:* [**Lieberman**] • *Page:* **379**

Color profile: Generic CMYK printer profile
Composite Default screen

playing module represents its actions in general terms, such as "Move any ace that is on top of a tableau pile to an empty foundation slot." The visual processing component maps this command to the specific state of the Solitaire application: "Move the ace of spades to the second foundation slot." VisSolitaire, like a human solitaire player, relies on the layout of the cards to guide its actions, rather than the visual representations of the cards alone. VisMap's recognition of cards is one illustration of an application-specific visual recognition procedure that can be used in visual generalization.

To make a visual recognition approach work for PBE in general, we may have to define visual grammars that describe the meaning of particular interface elements or the visual language of particular applications. For example, if we understand that the format of a monthly calendar is a grid of boxes, each box representing the date and lines within the boxes representing particular appointments, we can infer the properties of a Now Up-to-Date Appointment object.

It is also possible that other properties of the appointment object (e.g., the duration of the appointment) are not represented in the visual display, so we may not be able to infer them from the screen representation alone. Developing the application display format grammars is time-consuming work for expert developers, not for end users. However, the effort for a particular application can be amortized over all the uses of that application. The model of the application need not be complete; it may only capture those aspects of the application data of current interest.

One way to utilize the results of this kind of processing in a PBE system is to adopt a similar approach to Tatlin (Leiberman 1998), which infers user actions by polling applications for their state periodically and compares successive states to determine user actions. Tatlin used the examinability of the application data models for the spreadsheet Excel and calendar program Now Up-to-Date via the Applescript interprocess communication language. In the scenario in Figure 19.4, a user copies information from a calendar and pastes it into a spreadsheet. Tatlin "sees" that the data pasted into the spreadsheet are the same as were selected in the calendar and infers the transfer operation.
If we developed descriptions of the visual interface of the calendar and the spreadsheet, we could do the same simply by analyzing the screen image, even without access to the underlying application data.

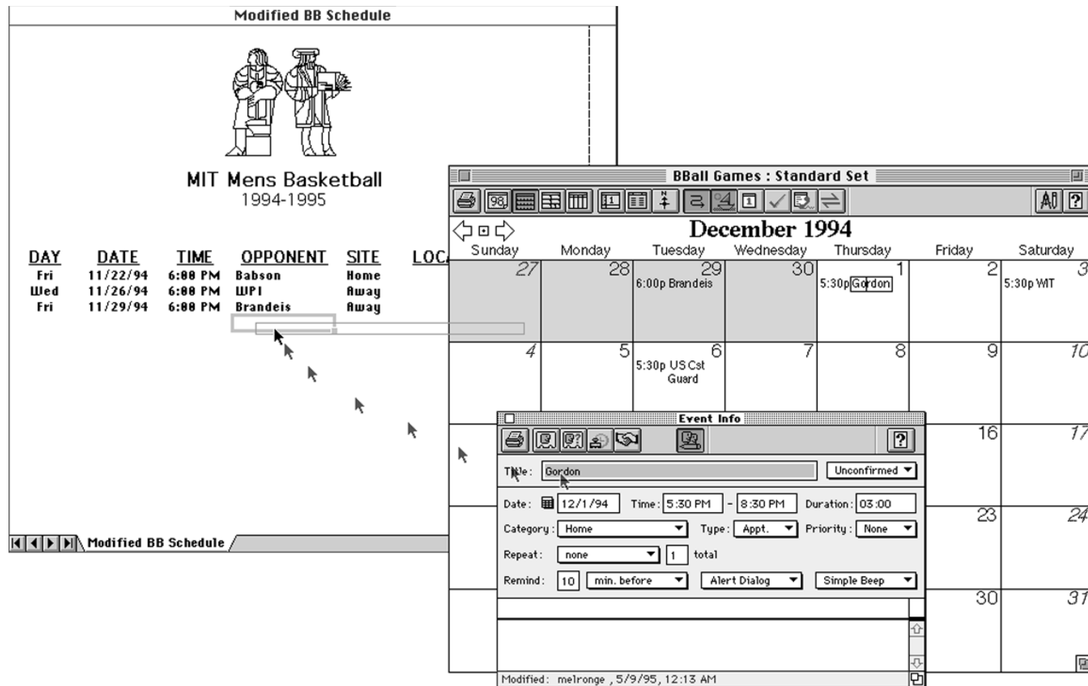
Research by others gives further evidence of the potential of visual generalization. Lakin (1987) built several programming environments around an object-oriented graphical editor, Vmacs. He used a recognition procedure on the visual relations between objects to attach semantics to sketched objects, which implemented a kind of visual generalization.

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:50 PM
TNT Job Number: [**002564**] • *Author:* [**Lieberman**] • *Page:* **380**

Color profile: Generic CMYK printer profile
Composite Default screen

FIGURE 19.4



Tatlin inferring copying data from a calendar to a spreadsheet.

Notably, the grammars used to drive the recognition procedure were themselves represented visually in Vmacs itself. Kurlander (1988) used a kind of visual generalization to automate search-and-replace procedures. But while Lakin and Kurlander were able to access the visual properties of objects directly in their own purpose-built graphical editors, we are proposing to extract the same kind of visual properties directly from pixel-level analysis of the screen.

19.6 Introducing Novel Generalizations: Generalizing On Grids

Visual generalization opens us the possibility of having different kinds of generalizations than are possible by generalizing from the properties of the underlying application data. As an example of a kind of useful

S
R
L

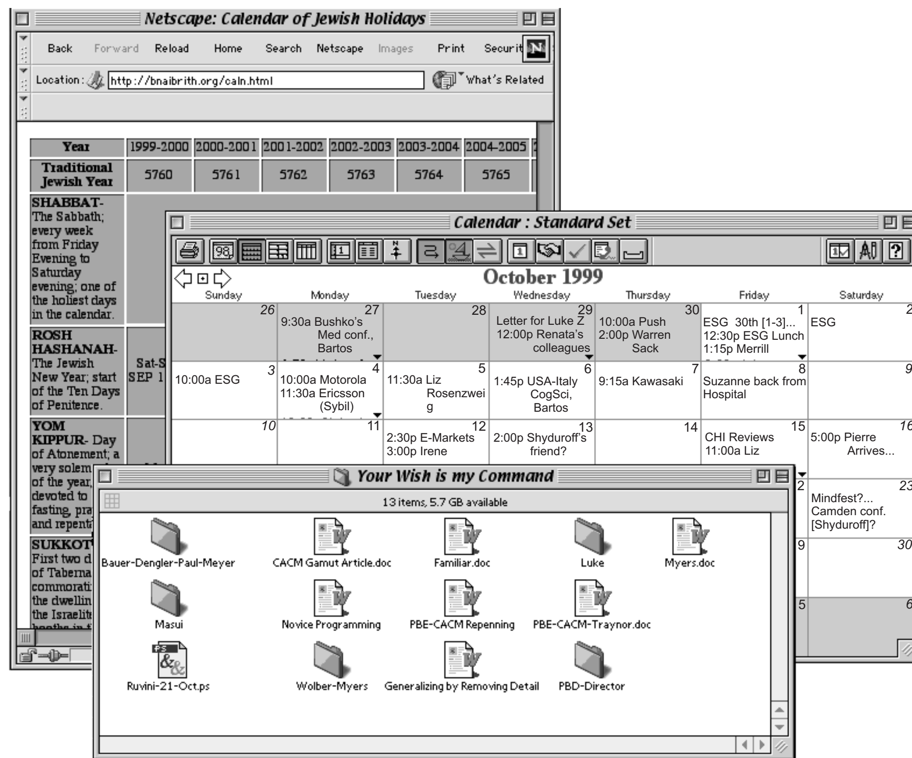
Color profile: Generic CMYK printer profile
Composite Default screen

generalization not possible with data-based approaches, consider that it might be possible to convey the general notion of a *grid,* so that procedures might be iterated throughout the elements of a grid (Figure 19.5).

The idea of a grid can be expressed purely with visual relations: "You start at one object, then move right until you find the next, and so on, until there are no more objects to the right. Then return to the object in the beginning of the row, move down one object, then start moving to the right again. Keep doing each row until you can't move down anymore."

Once you have the "idea" of a grid, you can apply it in a wide variety of applications. The same program could work whether operating on daily schedules in a calendar, program, icons in a folder window, or tables in Netscape, among other things.

FIGURE **19.5**



Examples of grids in (a) a Calendar, (b) the Finder, and (c) Netscape.

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:56 PM
TNT Job Number: [002564] • Author: [Lieberman] • Page: 382

Color profile: Generic CMYK printer profile
Composite Default screen

For this to work, the definition of "move to the next object to the left" and "move to the next object down" may need to be redefined for each application. But given the ability to do so, we can make real the user's perception that all grids are basically the same, despite the artificial barriers that separately programmed applications place against this generalization.

19.7 Conclusion

We can ask several questions when exploring a new perspective such as that offered by a visual generalization approach. For example, how can it contribute in a way that other existing perspectives are unable to? Existing techniques such as Apple Events and OLE Automation can sometimes provide powerful perspectives from which to build programs. Adding a new perspective to a system can increase the user interface complexity significantly. If there is a large overlap in the range of information, then the new form of the information must provide some advantage—as can be demonstrated with Triggers and VisMap.

What new challenges are raised by the new perspective, and what tools can address the challenges? Triggers has the challenge of accurately specifying pixel patterns and distances that are cryptic when viewed out of context. It addresses this challenge using the Desktop Blanket, a technique for allowing direct manipulation widgets to float above the screen pixels of the display. VisMap has the challenge of inferring high-level features from low-level pixel data. It addresses this challenge using a two-stage translation process. The first stage works bottom-up and identifies low-level features. The second stage works top down and infers high-level features from the low-level features.

Can complete solutions be built within the perspective? Such solutions may indicate the potential for an elegant special purpose system. Working from one perspective, it has potential to have a simple elegant interface. Triggers show a small set of functionality that can automate nontrivial tasks. More work has to be done, however, to show that a significant user group can make use of this functionality.

How can we integrate the perspective with other perspectives? The Triggers-IV system described in Potter (1999) addresses these issues by showing how the Desktop Blanket can be added to a conventional programming language. VisMap already has a textual interface that can easily be integrated with textual programming languages that use other techniques.

S
R
L

Color profile: Generic CMYK printer profile
Composite Default screen

Our current intuitions about the design of a visual generalization system for PBE lean toward a broad-based approach that applies pixel-level operators, as in TRIGGERS, where appropriate, but also generates higher-level information inferred from the pixel data, as in VisMap. If the user knows what a particular piece of information looks like on the screen but does not know how to describe it, then a low-level pixel based approach may be the best choice. If displayed information needed by a program is not provided by formal techniques and its visual appearance is complicated, then a high-level pixel-based approach may be the best solution. If the program needs efficient access to large data structures in an application, then the user can choose a technique such as OLE Automation or Apple Events, provided the application provides the necessary support.

Other issues for complete integration in applications include the granularity of event protocols, styles of interaction with the user, and parallelism considerations (Leiberman 1998). Event granularity determines the level of abstraction at which a visual system interacts with an interface. For example, should mouse movements be included in the information exchanged? If not all mouse movements, then which ones are important? Issues of parallelism can enter the picture when the system and the user both try to manipulate the same interface object.

We believe that the opportunities and challenges of visual generalization will be a fruitful new direction for PBE in the future. It might turn out that when it comes to graphical interfaces, beauty may indeed be only skin deep.

References

Halbert, Dan. 1993. Programming by demonstration in the desktop metaphor. In *Watch what I do: Programming by demonstration,* ed. Allen Cypher. Cambridge, Mass.: MIT Press.

Kurlander, D., and E. Bier. 1988. Graphical search and replace. In *Proceedings of ACM SIGGRAPH '88.*

Lakin, Fred. 1987. Visual grammars for visual languages. In *AAAI-87: The Conference of the American Assocation for Artificial Intelligence* (Seattle, Wash., July 12–17).

Lieberman, Henry. 1998. Integrating user interface agents with conventional applications. *Knowledge-Based Systems* 11, no. 1 (September): 15–24.

Potter, R. 1993. Triggers: Guiding automation with pixels to achieve data access. In *Watch what I do: Programming by demonstration,* ed. Allen Cypher. Cambridge, Mass.: MIT Press.

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:57 PM
TNT Job Number: **[002564]** • *Author:* **[Lieberman]** • *Page:* **384**

Color profile: Generic CMYK printer profile
Composite Default screen

———. 1999. *Pixel data access: Interprocess communication in the user interface for end-user programming and graphical macros.* Ph.D. diss. University of Maryland.

Yamamoto, K. 1998. A programming method of using GUI as API. *Transactions of Information Processing Society of Japan* (December): 26–33 (in Japanese).

Zettlemoyer, L., and R. St. Amant. 1999. A visual medium for programmatic control of interactive applications. In *Proceedings of ACM CHI'99 Human Factors in Computing Systems.*

S
R
L

V:\002564\002564.VP
Thursday, December 21, 2000 2:14:57 PM
TNT Job Number: [**002564**] • *Author:* [**Lieberman**] • *Page:* **385**