IPR No.: IPR2016-00500
Patent No. 7,864,163

EXHIBIT 1006

acquire the concept of *conservation* or *invariance*. At about age 11, children enter the formal-operations stage, in which they use *symbol manipulation* to represent actions on objects. Since mathematics and programming require abstract thinking, they are difficult for children, and designers must link symbolic representations to actual objects. Direct manipulation brings activity to the concrete-operational stage, thus making certain tasks easier for children and adults.

6.3.3 Visual thinking and icons

The concepts of a *visual language* and of *visual thinking* were promoted by Arnheim (1972) and were embraced by commercial graphic designers (Verplank, 1988; Mullet and Sano, 1995), semiotically oriented academics (*semiotics* is the study of signs and symbols), and data-visualization gurus. The computer provides a remarkable visual environment for revealing structure, showing relationships, and enabling interactivity that attracts users who have artistic, right-brained, holistic, intuitive personalities. The increasingly visual nature of computer interfaces can sometimes challenge or even threaten the logical, linear, text-oriented, left-brained, compulsive, rational programmers who were the heart of the first generation of hackers. Although these stereotypes—or caricatures—will not stand up to scientific analysis, they do convey the dual paths that computing is following. The new visual directions are sometimes scorned by the traditionalists as *WIMP* (windows, icons, mouse, and pull-down menu) interfaces, whereas the command-line devotees are seen by visual system proponents as stubborn and inflexible.

There is evidence that different people have different cognitive styles, and it is quite understandable that individual preferences may vary. Just as there are multiple ice-cream flavors or car models, so too there will be multiple interface styles. It may be that preferences will vary by user and by tasks, so respect is due to each community, and the designer's goal is to provide the best of each style and the means to cross over when desired.

The conflict between text and graphics becomes most heated when the issue of *icons* is raised. Dictionary definitions of *icon* usually refer to religious images, but the central notion in computing is that an icon is an image, picture, or symbol representing a concept (Rogers, 1989; Marcus, 1992). In the computer world, icons are usually small (less than 1-inch-square or 64- by 64-pixel) representations of an object or action. Smaller icons are often used to save space or to be integrated within other objects, such as a window border or toolbar. It is not surprising that icons are often used in painting programs to represent tools or actions (for example, lasso or scissors to cut out an image, brush for painting, pencil for drawing, eraser to wipe clean), whereas word processors usually have textual menus for their actions. This difference appears to reflect the differing cognitive styles of visually and textually oriented users, or at least differences in the tasks. Maybe while users are working on visually oriented tasks, it is helpful to "stay visual" by using icons, whereas while working on a text document, it is helpful to "stay textual" by using textual menus.

For situations where both a visual icon or a textual item are possible—for example, in a directory listing—designers face two interwoven issues: how to decide between icons and text, and how to design icons. The well-established highway signs are a useful source of experience. Icons are unbeatable for showing ideas such as a road curve, but sometimes a phrase such as ONE WAY!—DO NOT ENTER is more comprehensible than an icon. Of course, the smorgasbord approach is to have a little of each (as with, for example, the octagonal STOP sign), and there is evidence that icons plus words are effective in computing situations (Norman, 1991). So the answer to the first question (deciding between icons and text) depends not only on the users and the tasks, but also on the quality of the icons or the words that are proposed. Textual menu choices are covered in Chapter 7; many of the principles carry over to icon use. In addition, these icon-specific guidelines should be considered:

- Represent the object or action in a familiar and recognizable manner.
- Limit the number of different icons.
- Make the icon stand out from its background.
- Carefully consider three-dimensional icons; they are eye-catching but also can be distracting.
- Ensure that a single selected icon is clearly visible when surrounded by unselected icons.
- Make each icon distinctive from every other icon.
- Ensure the harmoniousness of each icon as a member of a family of icons.
- Design the movement animation: when dragging an icon, the user might move the whole icon, just a frame, possibly a grayed-out or transparent version, or a black box.
- Add detailed information, such as shading to show the size of a file (larger shadow indicates larger file), thickness to show the breadth of a directory folder (thicker means more files inside), color to show the age of a document (older might be yellower or grayer), or animation to show how much of a document has been printed (document folder absorbed progressively into the printer icon).
- Explore the use of combinations of icons to create new objects or actions—for example, dragging a document icon to a folder, trash can, outbox, or printer icon has great utility. Can a document be appended or prepended to another document by pasting of adjacent icons? Can a user set security levels by dragging a document or folder to a guard dog, police car, or vault icon? Can two database icons be intersected by overlapping of the icons?

Marcus (1992) applies semiotics as a guide to four levels of icon design:

1. *Lexical qualities.* Machine-generated marks—pixel shape, color, brightness, blinking
2. *Syntactics.* Appearance and movement—lines, patterns, modular parts, size, shape
3. *Semantics.* Objects represented—concrete versus abstract, part versus whole
4. *Pragmatics.* Overall legibility, utility, identifiability, memorability, pleasingness

He recommends starting by creating quick sketches, pushing for consistent style, designing a layout grid, simplifying appearance, and evaluating the designs by testing with users. We might also consider a fifth level of icon design:

5. *Dynamics.* Receptivity to clicks—highlighting, dragging, combining

The dynamics of icons might include a rich set of gestures with a mouse, touch-screen, or pen. The gestures might indicate copy (up and down arrows), delete (a cross), edit (a circle), and so on. Icons might also have associated sounds. For example, if each document icon had a tone associated with it (the lower the tone, the bigger the document), when a directory was opened, each tone might be played simultaneously or sequentially. Users might get used to the symphony played by each directory and be able to detect certain features or anomalies, just as we often know telephone numbers by tune and can detect misdialings as discordant tones.

Icon design becomes more interesting as computer hardware improves and as designers become more creative. Animated icons that demonstrate their functions improve online help capabilities (see Section 13.4.2). Beyond simple icons, we are now seeing increasing numbers of visual programming languages (see Section 5.3) and specialized languages for mechanical engineering, circuit design, and database query.

6.3.4 Direct-manipulation programming

Performing tasks by direct manipulation is not the only goal. It should be possible to do programming by direct manipulation as well, at least for certain problems. As mentioned earlier, people often program car-painting robots by moving the robot arm through a sequence of steps that are later replayed, possibly at higher speed. This example seems to be a good candidate for generalization. How about moving a drill press or a surgical tool through a complex series of motions that are then repeated exactly? In fact, these direct-manipulation programming ideas are already being implemented in modest ways with automobile radios that users preset by tuning to their desired station and then pressing and holding a button. Later, when the button is pressed, the radio tunes to the preset frequency. Likewise, some professional television-camera supports allow the operator to program a sequence of pans or zooms and then to replay it smoothly when required.

Programming of physical devices by direct manipulation seems quite natural, and an adequate visual representation of information may make direct-manipulation programming possible in other domains. Several word processors allow users to create macros by simply performing a sequence of commands and storing it for later use—for example, emacs allows its rich set of functions, including regular-expression searching, to be recorded into macros. Spreadsheet packages, such as Lotus 1-2-3 and Excel, have rich programming languages and allow users to create portions of programs by carrying out standard spreadsheet actions. The result of the actions is stored in another part of the spreadsheet and can be edited, printed, and stored in a textual form. Similarly, Adobe PhotoShop records a history of user actions and then allows users to create programs with action sequences and repetition using direct manipulation.

It would be helpful if the computer could recognize repeated patterns reliably and create useful macros automatically, while the user was engaged in performing a repetitive interface task. Then, with the user's confirmation, the computer could take over and carry out the remainder of the task automatically (Lieberman, 2001). This hope for automatic programming is appealing, but a more effective approach may be to give users the visual tools to specify and record their intentions. Rule-based programming with graphical conditions and actions offers a fresh alternative that may be appealing to children and adults (Smith, Cypher, and Spohrer, 1994). The screen is portrayed as a set of tiles, and users specify graphical rewrite rules by showing before-and-after tile examples (Fig. 6.11). Another innovative environment initially conceived of for children is ToonTalk (Kahn, 1999), which offers animated cartoon characters who carry out actions in buildings using a variety of fanciful tools.

To create a reliable tool that works in many situations without unpredictable automatic programming, designers must meet the five challenges of *programming in the user interface* (PITUI) (Potter, 1993):

1. Sufficient computational generality (conditionals, iteration)
2. Access to the appropriate data structures (file structures for directories, structural representations of graphical objects) and operators (selectors, booleans, specialized operators of applications)
3. Ease in programming (by specification, by example, or by demonstration, with modularity, argument passing, and so on) and in editing programs
4. Simplicity in invocation and assignment of arguments (direct manipulation, simple library strategies with meaningful names or icons, in-context execution, and availability of results)
5. Low risk (high probability of bug-free programs, halt and resume facilities to permit partial executions, undo actions to enable repair of unanticipated damage)

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.