Sketching intelligent systems

By Marti A. Hearst
University of California, Berkeley
hearst@sims.berkeley.edu

Those who have been reading this feature on a regular basis no doubt have noticed my penchant for topics on human-computer interaction. An important general trend in intelligent systems is that of acknowledging the smarts of the users of our systems, thus complementing efforts to put smarts into the systems themselves. The growing importance of HCI is underscored by the naming of Douglas Engelbart as last year's recipient of the prestigious ACM Turing Award. Engelbart was also showered with honors at this year's ACM conference on human computer interaction (SIGCHI) in Los Angeles (*http://www.acm.org/sigchi* and *http://www.acm.org/pubs/contents/proceedings/chi/*). A trend that surfaced at this meeting was the move toward more natural, human-centered forms of interaction. Many presentations explored alternatives to the traditional keyboard, mouse, and monitor, replacing these with tangible physical artifacts. Examples included embedding of interfaces into an easy chair, a ping-pong table, name tags, children's plastic beads, and triangular shapes that store audio clips, which can be used to create stories based on which sides of the triangles are snapped together.

In this installment of Trends and Controversies we look at another aspect of this trend toward more natural forms of interaction: the use of sketch-based interfaces for the design of intelligent systems.

Our authors contend that current computer interfaces are too formal and precise for creative tasks such as design. When working out ideas and brainstorming, people often sketch their thoughts informally on paper and whiteboards. Sketches, by their very informalness, invite collaboration and modification. But after sketches are done on paper they must be transferred to the computer for further processing.

Why not use a computer for the sketching process itself? As Mark Gross points out below, there is an inherent contradiction in today's sketching programs- -users select graphical primitives from a palette, resulting in clean, precise-looking diagrams despite the fact that only a sketch was intended. This leads to a feeling of commitment to the sketch as originally formulated, as opposed to the invitation to adjust and change that is normally associated with sketches.

By contrast, this month's authors (Mark Gross, James Landay, and Tom Stahovich) describe sketch-based user interfaces that honor the informal nature of the sketch and allow users to switch easily and naturally between sketchy, informal prototypes, and precise, formal reifications of designs. In these pages, sketches are used for the design of user interfaces (Landay), mechanical devices (Stahovich), and web pages (Gross and Landay), to retrieve images from a large collection (Gross), and to specify parameters for simulations (Gross).

Stahovich points out that a sketch is useful as a reasoning tool because it presents a particular example of a problem to think about as a starting point, when the general case is too abstract and elusive. In Gross's image-retrieval system, the sketch is an example of the kinds of images of interest. In Landay's user-interface design system, the sketch is an example of the functionality and basic interaction styles of the interface to be built. In Stahovich's mechanical-device-design system, the sketch is just one example from which the system infers an entire family of designs.

Although a sketch produces a concrete example of a family of designs, its informal nature somewhat paradoxically also allows details to be left unspecified. (This is why a sketch is different from other ways of specifying examples.) Landay points out the utility of the ambiguity of a sketch, which makes it possible for a designer to delay decisions about details until an appropriate stage in the design process.

The systems described in the essays use AI techniques to convert sketches to their more precise computerized counterparts. Gross's Electronic Napkin uses symbol- and configuration-recognition techniques to define a visual language grammar. Landay's Silk uses a statistically-based gesture recognizer and a rule system to combine primitive components into more complex ones, and dynamic storyboarding, which allows users to define user intervals that contain both standard widgets and novel components. Stahovich's SketchIT uses qualitative-reasoning techniques to identify mechanical interactions between the parts of the sketch of a device and a state-transition diagram of its desired behavior.

All three authors emphasize the importance of using the most appropriate input modality for the task at hand. How to automatically determine what tools to offer when is an open question. Nevertheless, the work presented here represents real progress towards the the goal of natural, humanistic interaction with computer systems.

—Marti Hearst

The proverbial back of an envelope

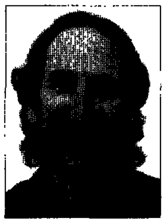
Mark D. Gross, Department of Planning and Design, University of Colorado

People draw diagrams and sketches to think, argue, and communicate about problems in domains ranging from mechanical engineering to music. To understand a problem better, mathematician George Polya advised, "Draw a figure ... even if your problem is not a problem of geometry."[1] We can debate whether making a figure really helps people solve problems (and if so, how); nevertheless, we draw diagrams all the time. Almost every office, laboratory, and classroom offers a whiteboard where people draw informal diagrams to illustrate their ideas, and the napkin sketch and the back of an envelope are proverbial design tools. Especially for design applications, it makes sense to try to construct interactive systems that can accept, parse, and recognize this common mode of human communication as an input modality.[2]

Despite the prevalence of "graphical" user interfaces, we interact with computer tools for the most part by choosing from menus, pressing buttons, and entering text in forms and files. The first interactive graphics systems, notably Sketchpad,[3] employed light pens that let users point and mark directly on a cathode-ray tube image. But with raster graphics in the mid-1970s came a widespread adoption of the mouse, relegating the stylus to relative obscurity. This persisted until the early 1990s when—with the advent of pen-based computers and personal digital assistants—the computer industry suddenly rediscovered the stylus. Even today, most PDA and pen-based applications focus on input of text, not graphics.

To be useful, the machine needn't be able to make sense of Picasso's sketches: most whiteboard drawings are highly stylized or diagrammatic. They are made up of graphical symbols selected from a fairly small universe, arranged in a fairly small number of spatial relationships and often augmented by text labels. But if most drawings are diagrams, why not make them using a structured draw program interface, selecting graphical primitives from a palette and assembling them into a diagram instead of drawing them? Indeed, many "sketch" programs require the user to do exactly that, and the resulting diagrams are clean and precise. Two arguments spring to mind against this approach. First is direct manipulation: why use a menu when you can just draw what you want? Second is the objective: in design, you might prefer to work with a crude or even ambiguous drawing.

Most systems that recognize hand-drawn graphics boast a "beautify" feature—the machine converts sloppily drawn figures to perfect ones. Yet the purpose of the informal sketch argues for leaving it exactly as drawn. Beautifying just elevates its precision and apparent commitment to a level that the drawer probably did not intend. A crudely drawn sketch implicitly indicates that the scale cannot be trusted and details have been left out. But if it is beautified, a reader might be tempted to assume that it is drawn to scale. Also, especially in design, it might be preferable to leave parts of the drawing unresolved, to be determined later. For these reasons, imprecision and even crudeness are valuable in early design drawings.



Mark D. Gross is an associate professor in the University of Colorado's Department of Planning and Design and also a member of its Institute for Cognitive Science. His research interests include computer support for design processes, coordination of team design work, and human-computer interfaces. He received a BS in architectural design and a PhD in design theory and methods, both from MIT. He is a member of the IEEE, ACM, AAAI, and the Association for Computer Aided Design in Architecture. Contact him at the Sundance Laboratory, Univ. of Colorado, Boulder CO, 80309-0314; mdg@spot.colorado.edu; http://spot.colorado.edu/~mdg/.

James A. Landay is an assistant professor in the Computer Science Division of the EECS Department at the University of California at Berkeley. His research interests include human-computer interaction, user interface design tools, pen-based user interfaces, end-user programming, mobile computing, and other novel uses of computer technology. He earned a BS in electrical engineering and computer science from Berkeley and an MS and PhD in computer science from Carnegie Mellon University. He is a member of the ACM, SigChi, and SigGraph. Contact him at the Computer Science Div., Univ. of California at Berkeley, Berkeley, CA, 94720-1776; landay@cs.berkeley.edu; http://www.cs.berkeley.edu/~landay.

Thomas F. Stahovich is an assistant professor in the Department of Mechanical Engineering at Carnegie Mellon University. His research interests focus on designing and building intelligent software tools to support mechancial engineering design. He received a BS from the University of California at Berkeley and an SM and PhD from MIT, all in mechanical engineering. He is a member of the AAAI and the ASME. Contact him at the Dept. of Mechanical Engineering, Carnegie Mellon Univ., Pittsburgh. PA 15213; stahov@andrew.cmu.edu; http://www.me. cmu.edu/faculty1/stahovich/stahovich.html.

Electronic Cocktail Napkin

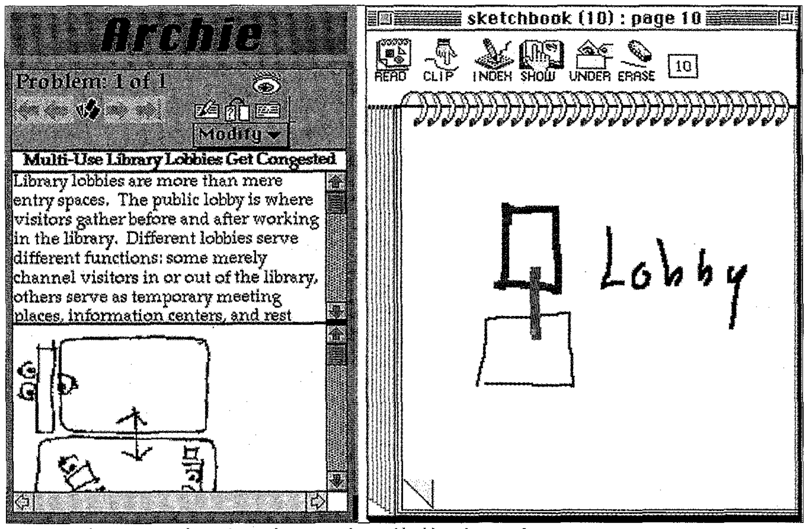
To explore these ideas, my students and I have been working on a project we call the Electronic Cocktail Napkin (and its successor, the Back of an Envelope). We aim to develop a freehand drawing environment that can serve as an interface to a variety of applications that support designers, such as case-based advisors, critics, simulations, libraries, and (for architects and engineers) drafting and modeling software.

We have observed that many architects, even those fluent with computer-aided design software, choose to develop their initial ideas using paper and pencil, and only translate their sketches and drawings to electronic media for design development once the conceptual design work has been done. This way of working imposes a time-consuming discontinuity in the design process between conceptual exploration and design development that is largely irreversible: you cannot move back and forth easily between sketch and computational representation. As a consequence, computer-based tools that might be useful to designers are unavailable in the earliest and most formative stages of design. They can only be employed during design development after the designer has made the most critical decisions.

The Electronic Cocktail Napkin supports freehand drawing on a digitizing tablet, a whiteboard, and a PDA, and it allows several users drawing on different devices to work together on a drawing. It simulates various drawing instruments (pencil, pen, marker, brush) and media (tracing paper, sketchbook). At first glance, the program appears merely to combine the features of a simple paint program (users draw freehand) and a drawing program (they select, resize, move, and delete elements). But beyond emulating physical media and conventional drawing program software, the Electronic Cocktail Napkin also provides trainable symbol recognition, parses more complex configurations of symbols and spatial relations, and can match similar figures.

The Napkin supports recognition of simple glyphs, or symbols. It delimits individual symbols by the length of time the pen is lifted; a symbol need not be restricted to a single stroke. It recognizes a symbol by comparing its features—pen path, number of strokes and corners, and aspect ratio—with a library of stored templates. Each user works with a personal library of glyphs, which allows idiosyncratic drawing styles. The Napkin supports on-the-fly training: a user adds to the library of templates just by drawing a few examples and identifying the symbol.

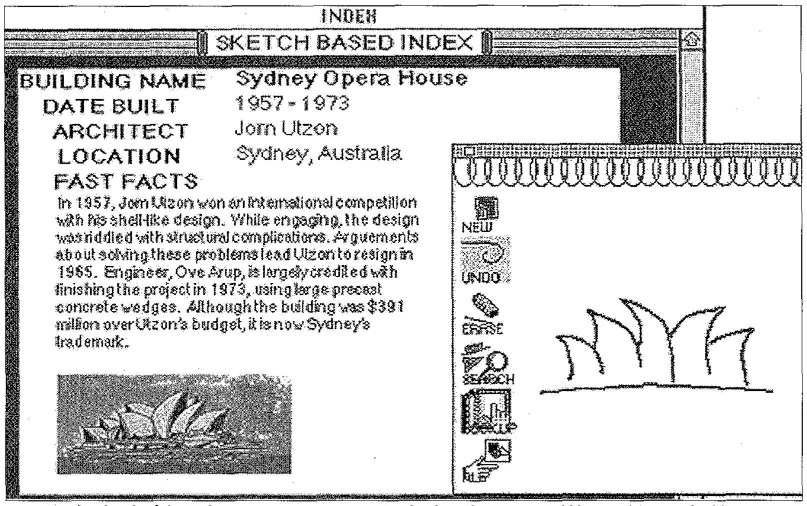
The Napkin identifies spatial relationships among the elements of a diagram—for example, left-of, above, contains, or connects. It recognizes user-defined con-



Multi-Use Library Lobbies Get Congested

Library lobbies are more than mere entry spaces. The public lobby is where visitors gather before and after working in the library. Different lobbies serve different functions: some merely channel visitors in or out of the library, others serve as temporary meeting places, information centers, and rest

Figure 1. A diagram is used to index Archie, a case base of building design information.



BUILDING NAME	Sydney Opera House
DATE BUILT	1957 - 1973
ARCHITECT	Jorn Utzon
LOCATION	Sydney, Australia

FAST FACTS

In 1957, Jorn Utzon won an international competition with his shell-like design. While engaging, the design was riddled with structural complications. Arguements about solving these problems lead Utzon to resign in 1965. Engineer, Ove Arup, is largely credited with finishing the project in 1973, using large precast concrete wedges. Although the building was $391 million over Utzon's budget, it is now Sydney's trademark.

Figure 2. The sketch of the Sydney Opera House serves as a bookmark into a visual library of famous buildings.

figurations of elements arranged in certain spatial relationships; for example, a tree diagram defined as "circles or rectangles, one vertically above the other, connected by line segments." A user defines a configuration recognizer by drawing examples; the Napkin offers a description of the configuration, and the user refines the description. As with simple glyphs, training a new configuration takes place on the fly, not as a separate programming step.

With these two recognition capabilities (symbols and configurations) the Electronic Cocktail Napkin lets users graphically and interactively define a visual-language grammar, which the program can then use to parse freehand drawing input. In addition to parsing compound configurations, the Napkin can also diagnose the similarity of two diagrams, comparing the numbers and types of elements and the spatial relations in each.

Applications

Our program has served as a platform to build prototype interfaces to a variety of applications. We have built a visual indexing and retrieval scheme, a freehand drawing interface to simulation programs, an HTML layout application, and a simple constraint-based diagram editor.[4,5]

Visual bookmarks. We used the Napkin's similarity matching to develop a visual bookmarking scheme. A designer, browsing a digital library of images or design information, makes a diagram to index an item of interest. Later, drawing a similar diagram retrieves the database entry or image. We built a diagram index for a case library of architectural design problems and solutions (see Figure 1), for a published CD-ROM library of famous buildings (see Figure 2), and for URLs in the World Wide Web. In each case, the designer draws a diagram on the Napkin's sketchbook page, which also stores linking information for various external databases and applications. When the designer draws a diagram to index a Web page, Napkin asks the browser for the current URL and stores that address with the diagram in the sketchbook. To retrieve the page, after finding a matching diagram in the sketchbook the program sends the browser a "go to URL" message.

Interface to simulations. We used the Napkin as an interface to drive two interactive simulation tools. In the first example, using a local area network design tool called ProNet, the designer draws a freehand diagram of a network to be simulated; the Napkin parses the diagram and sends instructions to ProNet to construct and run the simulation. The second simulation tool, IsoVist, helps architects compute and visualize the *isovist*, or territory visible from a given vantage (see Figure 3). The designer draws the floor plan on the Napkin, which parses and exports it to the IsoVist calculator, which performs the IsoVist calculation and displays the result.

HTML editing. Our WebStyler prototype explored using freehand diagrams to lay out a Web page. The designer draws the page layout using graphic symbols to position titles, text, and pictures. After the designer attaches specific text and pictures to the page, WebStyler generates appropriate HTML code to position the items correctly in the layout (see Figure 4).

Discussion

The same diagram might have rather different meanings in different domains. In surveying whiteboards at my university, we found that across a wide range of departments the diagrams contained mostly the same symbols and spatial relations, although they quite clearly were talking about different subjects. Context plays a large role in forming a correct interpretation of a diagram, as it does in natural-language understanding. For example, a curlicue line means "inductance" in the context of analog circuits, but in a mechanics diagram it means "spring," and in a child's drawing of a person, "hair." It's sometimes possible for the Napkin to recognize context by a unique symbol or configuration that appears in the diagram (for example, a treble clef indicates that the diagram is about music). Once the context is recognized, interpretation of symbols might change (curlicues become inductances or springs), as well as the set of recognizable configurations.

We're currently redesigning the Electronic Cocktail Napkin interface so that designers can use it as a tracing paper overlay on top of other applications. In this interaction mode, they would see through the Napkin's window and interact with the application by drawing freehand marks and diagrams. For example, an architect could interact with a modeling program, sketching on top of the 3D forms to edit them. Or the architect could edit a text document the same way you work on paper, using a pen to insert, delete, and move sections of text. This will require specifying a standard way of translating marks made on the Napkin to commands for the back-end application. It will also require enabling the back-end application to take control of the user's diagram.

In the Right Tool at the Right Time project, we're looking at whether the interface might be able to identify what the designer is doing just by looking at the drawing, and then provide an appropriate tool for the task at hand. For example, observing that an architect is drawing light rays and computer monitors, the Right Tool manager might proffer a lighting simulation program or advice in a case base about how to avoid glare in work areas.

Drawing is most often used in conjunction with other communication modalities. A drawing doesn't stand alone, but is embedded in a social context. The whiteboard drawing, the back-of-the-envelope diagram, and the napkin sketch are part of a conversation. We don't expect that interacting with computers will be any different in this regard than with people: the real payoff will come by integrating freehand drawing interfaces with other interaction modes, including conventional structured interfaces, speech, and text. As a first step in this direction, we capture and store the spoken conversation made during drawing, and tag the audio track with the elements of the drawing. The designers can touch an element in the drawing and play back the conversation that was happening when the element was drawn or referenced.

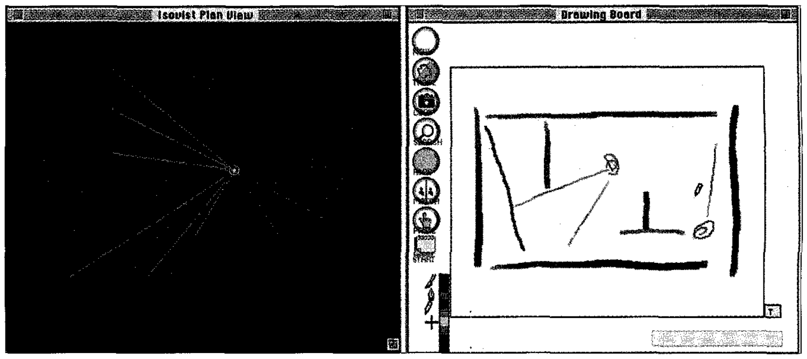
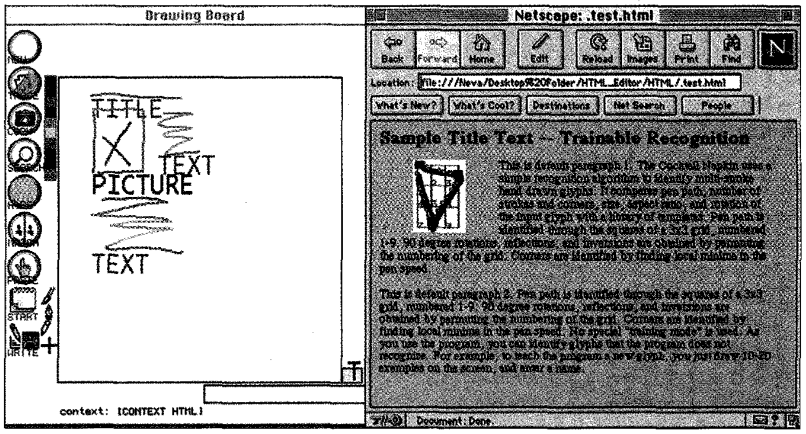


Figure 3. The Napkin sketch at right drives the IsoVist simulator at left, which computes and displays the viewshed in a floor plan.



Location: file:///Neva/Desktop%20Folder/HTML_Editor/HTML/.test.html

Sample Title Text — Trainable Recognition

This is default paragraph 1. The Cocktail Napkin uses a simple recognition algorithm to identify multi-stroke hand drawn glyphs. It compares pen path, number of strokes and corners, size, aspect ratio, and rotation of the input glyph with a library of templates. Pen path is identified through the squares of a 3x3 grid, numbered 1-9. 90 degree rotations, reflections, and inversions are obtained by permuting the numbering of the grid. Corners are identified by finding local minima in the pen speed.

This is default paragraph 2. Pen path is identified through the squares of a 3x3 grid, numbered 1-9. 90 degree rotations, reflections, and inversions are obtained by permuting the numbering of the grid. Corners are identified by finding local minima in the pen speed. No special "training mode" is used. As you use the program, you can identify glyphs that the program does not recognize. For example, to teach the program a new glyph, you just draw 10-20 examples on the screen, and enter a name.

Figure 4. WebStyler. The Web page at right was laid out using the Napkin sketch at left.

There's a chicken-and-egg problem here. It's not clear that an informal interface to a formal application is a good idea: if an application requires precision, freehand diagrams might be the wrong approach. Today's structured interfaces to design applications affect the role we expect the software to play, and thus the design of the software itself. For example, most energy-estimating tools for architects require precise input and deliver precise output. They are therefore less useful in the early stages of design, although that is when the design is most malleable. Perhaps the advent of informal interfaces will foster a new generation of sketchy applications, software that is more qualitative and less demanding of precision, and thereby more useful during early, conceptual design.

Acknowledgments

Support came from NSF grant IRI 96-19856 (the Back of an Envelope), and from a grant (with Wayne Citrin and Steve Laufmann) by the Colorado Advanced Software Institute and USWest Advanced Technologies. Ellen Yi-Luen Do, Adrienne Warmack, Kyle Kuczun, Jen Lewin, Paul Hamill, and Kristin Anderson made valuable contributions to various parts of the project. Thanks also to the L^{3}D group at the University of Colorado for the use of ProNet and to Janet Kolodner, Eric Domeshek, and Craig Zimring at Georgia Tech for the use of Archie.

References

1. G. Polya, *How to Solve It*, Princeton Univ. Press, Princeton, N.J., 1945, p. 97.
2. D. Ullman, S. Wood, and D. Craig, "The Importance of Drawing in the Mechanical Design Process," *Computer Graphics*, Vol. 14, No. 2, 1990, pp. 263–274.
3. I.E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," *Proc. Spring Joint Computer Conf.*, Amer. Fed. Information Processing Socs., Montvale, N.J., 1963, pp. 329–346.
4. M.D. Gross and E.Y.-L. Do, "Ambiguous Intentions," *Proc. ACM Symp. User Interface Software and Technology (UIST '96)*, ACM Press, New York, 1996, pp. 183–192.
5. K. Kuczun and M.D. Gross, "Local Area Network Tools and Tasks," *ACM Conf. Designing Interactive Systems 1997 (DIS '97)*, ACM Press, 1990, pp. 215–221.