Recognizing and Interpreting Diagrams in Design

Mark D. Gross
Division of Environmental Design and Institute of Cognitive Science
University of Colorado, Boulder, Colorado 80309-0314
mdg@cs.colorado.edu

ABSTRACT

Hand drawn diagrams are essential tools for thinking and communicating in the early phases of design, yet computer based drawing tools support diagramming and sketching only poorly. Key components of computational support for early design include recognition, interpretation, and management of diagrams The paper describes the motivation for, implementation of, and initial experience with the "computer as cocktail napkin" project, a design environment based on diagrams. It explains low level recognition of glyphs, construction of higher-level recognizers, and routines for managing diagrams in the cocktail napkin prototype.

1. INTRODUCTION

In the early phases of design in almost every domain diagrams play a key role. Ideas tend to be vague and commitments low, unsuited to and unworthy of the effort and precision that most computer-based drawing tools demand. Instead, design arguments unfold on cocktail napkins and the backs of envelopes where the medium supports and reflects the rough and quick quality of thought. The "computer as cocktail napkin" project aims to support early design in an environment for recognizing, interpreting, and managing hand-drawn diagrams and sketches. Designers need to sketch and diagram, at least in the early stages of designing; they will do it with or without a computer.

This project aims to build programs that can recognize the elements of a diagram and interpret them in the context of particular design domains such as building floor plans, mechanical devices, or electric circuits. This would enable a designer to apply computational tools for symbolic and numerical analysis such as critiquing, simulation, visualization, and retrieval of relevant cases. Without a diagram interpreter, these tools must wait until the design has developed to where it is worth the effort of entering it into the machine. By providing a computational environment for early design -- in the way that designers are accustomed to doing it -- we will be able to bring these tools to bear at a time when they can have the least cost and the most impact.

The cocktail napkin project falls at the intersection of four diverse research areas: sketch recognition and parsing of visual languages [9, 10, 20, 21] diagrammatic representations in problem-solving [3], collaborative work and shared drawing spaces [2, 12], and intelligent computer aids for design [6]. Here we focus on recognition and interpretation of hand-drawn input and its application in a computer environment for design, touching only briefly on the other topics.

Sketch recognition and inference was a major focus of the MIT Architecture Machine Group in the early 1970's [17], yet many questions raised there -- e.g. integration of symbols and sketches, interpretation of speed and pressure data and line overtracing, use of tracing paper -- remain unexplored. Lakin's "electronic design notebook" and his work on **vmacs** has identified key issues for visual language in design, for example, the inappropriateness of syntax-driven drawing, the performance nature of early design, and the inevitable demand for symbolic processing later in designing [15]. Futrelle's group also deals with machine recognition of diagrams; however they are more interested in formal diagrams in published technical articles and not the hand-drawn diagrams made during designing [5].

Other algorithms for recognizing the low-level glyphs have been developed [1, 18]; the chief advantages of the technique described here are that it is trainable, handles multi-stroke glyphs, and is simple to implement. The approach taken here diverges from the notion of "visual language" employed by Wittenburg and Weitzman [20], Golin, [9] and Helm [10] in that those approaches assume that input data will be a well-formed diagram in a grammatical, hence parsable, visual language (although Wittenburg’s predictive parsing approach allows the recognition of partially drawn diagrams). Similarly, Zhao's hand-drawn diagram recognizer [21] employs a two-tier recognition strategy like ours, but is concerned with input for syntax-directed editors. Although we hope to extract parts of diagrams that match certain syntactic descriptions, we cannot expect design diagrams to be "well-formed". Therefore the cocktail napkin approach tries to take advantage of recognition techniques used in parsing visual languages, while allowing that diagrams in design may be intentionally ambiguous, arbitrary, and vague.

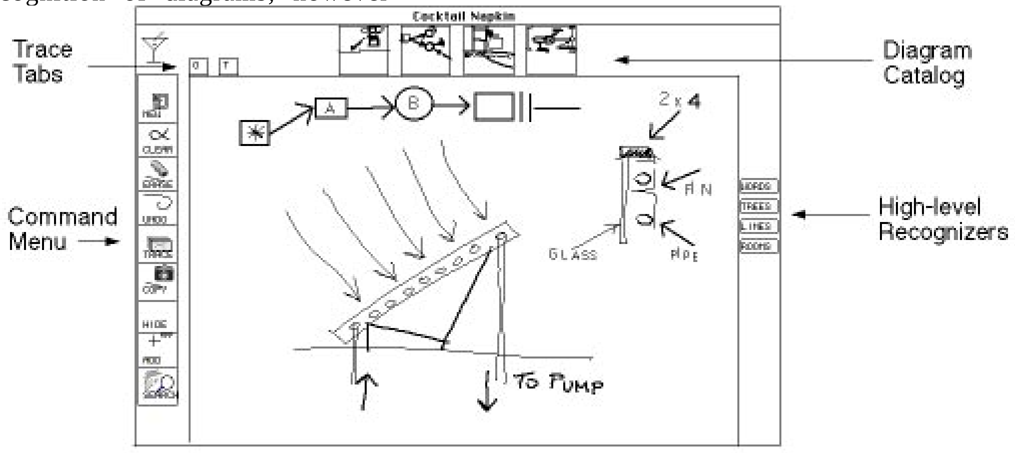


Figure 1: The cocktail napkin prototype.

The assertion that diagrams are more appropriate for early design is based largely on tradition (it's been done that way for centuries), and intuition (it makes sense). However, some evidence supports the claim. A well-known article by Larkin and Simon explored advantages of diagrammatic representations for certain kinds of reasoning [16]. They argue that diagrams rely on "perceptual inferences, which are extremely easy for humans," grouping information that is to be used together, thus limiting both the search for relevant information and the need for symbolic labels. In another study, Goel compared designers using a pencil with designers using MacDraw to carry out tasks in graphic design, industrial design, and architectural design [7]. Goel found that designers using a pencil generated more diverse alternatives whereas those using the structured drawing program tended to latch on early to a single alternative and develop it. Goldschmidt, studying the use of sketches in architectural design, argues that a serial sketching process is an essential part of design problem solving [8].

Figure 1 shows the screen of our cocktail napkin prototype, a diagram recognizer built in Macintosh Common Lisp to explore these ideas. It uses a WACOM "SD" series digitizing tablet and pen for input and it works interactively in real time on the Powerbook 160, MacIIfx, and Quadra models. The following sections of the paper describe first the low-level recognition of drawn symbols and spatial relations among them; second, how a designer constructs higher-level recognizers that identify arrangements of symbols satisfying a given set of spatial relations; and finally, some initial efforts toward managing the hand drawn diagrams.

2. RECOGNITION

2.1 Low Level Recognition

The low-level recognizer has been trained with a set of commonly used components of diagrams (e.g. circles, boxes, arrows, letters) and it identifies input glyphs by matching them against its training set. Most diagrams seem to be made from a small universe of symbols* so recognition of low-level glyphs need not resolve among a large number of candidates. The algorithm recognizes symbols in real time from a library of about 50. The low level recognizer proceeds in two steps. First, a raw-glyph parser examines coordinate and pressure data input from the pen and identifies strokes, corners, and shape. Second, the resulting profile is matched against a stored library of known glyphs.

* An informal survey of approximately 50 blackboards in our university's departments of chemistry, physics, mechanical engineering, geography, and architecture lends weight to this hunch.

The tablet driver delivers a sequence of x,y points and pressures sampled from the tablet at 2400 baud, as well as pen-up/down data. A glyph is deemed finished when the pen is off the pad for longer than 1/5 second. (This and other recognition parameters can be adjusted by the user; however if the end-of-glyph time-out is less than about 1/8 second it is difficult to execute multi-stroke glyphs.) If more than one pen is used (as when two users share the drawing pad) the tablet driver also identifies which pen was used to make the glyph.
The raw-glyph parser examines the tablet input data. It counts strokes and it identifies corners where the pen slowed down and the tablet sampled several points close together -- this also finds line crossings. The drawing sequence information makes trivial certain tasks, such as grouping together the several strokes of a single glyph, that are much more difficult in off-line analysis of finished drawings.

The raw-glyph parser identifies the input glyph's shape by overlaying a 3x3 grid with squares numbered 1-9 on its bounding box, determining the sequence of grid squares that the pen moved through to draw the glyph. This sequence is used as a lookup key in a hash table of previously trained glyphs. The 3x3 grid seems optimal for the shape match. A 2x2 grid is too coarse and results in many collisions; 4x4 is too fine and requires large training sets. Obviously, stroke order is essential to this scheme, and the program must be trained, or at least the training must be adjusted, on a per-user basis. To work for all users the program must be trained to recognize the various ways each glyph might be drawn. But van Sommers' studies of drawing and cognition suggest that different drawers show remarkable consistency in stroke order [19], so idiosyncrasy in drawing simple glyphs may not be a serious problem.

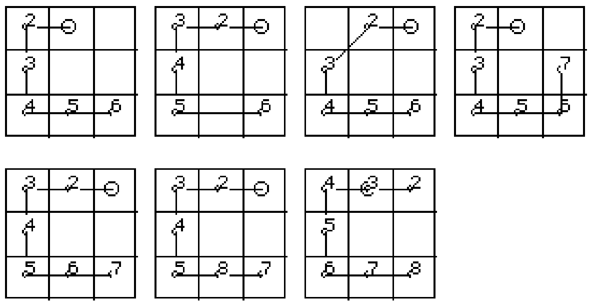


Figure 2: Training set for the "letter-C." The circle indicates the glyph's starting point. The

grid is scaled to the bounding box of the input glyph.

The 3x3 grid is inscribed in the input glyph's bounding box, so the algorithm works for all sizes and aspect ratios. It recognizes a tall, narrow A as well as a short, wide one. By permuting the numbering scheme of grid squares the algorithm can recognize 90 degree rotations, reflections of glyphs, and glyphs drawn backwards. Rotation between 90 degree multiples is covered by training the program with glyphs drawn at an angle. Figure 2 shows a set of grid square sequences for the glyph "letter-C," indicating seven different ways the program has seen that glyph drawn.

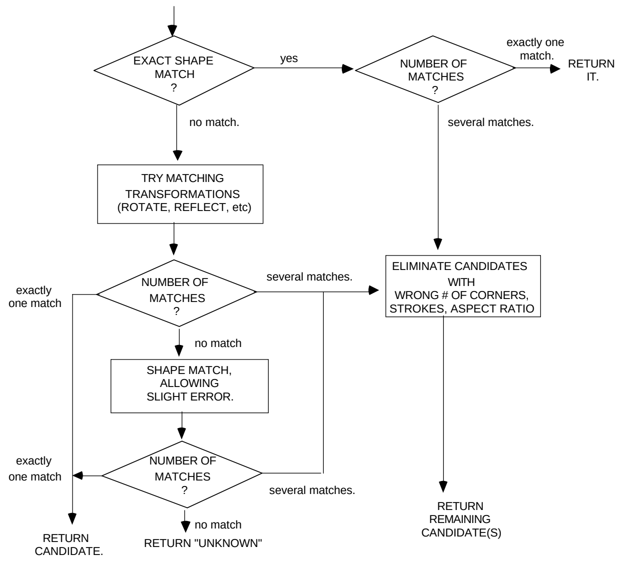


Figure 3: Steps in the low level recognizer.

Often two or more glyphs have been trained with the same sequence of grid squares and the shape lookup returns more than one candidate. For example, the sequence of grid squares may not distinguish 'U' from 'V' or 'Circle' from 'Box'. Then, a second low-level recognition step attempts to resolve the ambiguity by matching the number of strokes, corners, sizes, aspect ratios, and rotations allowed for each glyph. On the other hand, if the initial shape lookup yields no match, the program relaxes its criteria, allowing a small difference in the sequence of grid squares that describes the shape. The flow chart in figure 3 summarizes the matching procedure. The low level recognizer produces an "identified glyph" structure, shown in figure 4.

```
GLYPH
   :NAME ARROW.732
   :TYPE ARROW
   :NSTROKES 1
   :NCORNERS 4
   :SHAPE-SQUARES-SEQ (7 5 2 3 2 3 6
   :SIZE MEDIUM
   :ASPECT-RATIO SQUARE
   :ROTATION NORMAL
```

Figure 4: The data structure for an identified glyph includes stroke and corner count, pen path, and other information.

Even using the strategies described above, the recognizer may be unable to identify a glyph (for instance if the glyph has not been trained) or it may be unable to determine which of several glyphs in its training set best matches the new glyph. If it cannot identify the glyph, the low-level recognizer returns "no match"; if it cannot choose, it returns a list of candidates. Depending on switches the program either leaves ambiguous and unknown glyphs until later or it asks the user to immediately identify the glyph or resolve the ambiguity. Whenever the user resolves an ambiguity or names an unknown glyph, the program adds the new example to its training set. In general, to train the program the user draws examples of a new glyph and enters a name. The average glyph's training set has 13 samples; the largest set (for the letter A) contains 73 samples.

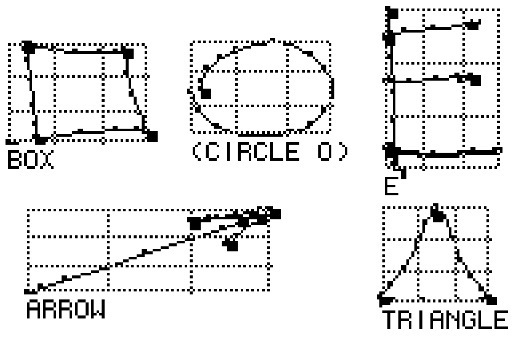
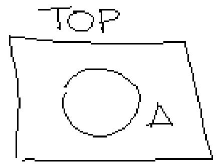


Figure 5: Raw glyphs, showing input points, identified corners, and the scaled 3x3 bounding box grid. The low level recognizer, unable to determine the top center glyph a circle or the letter "O," returned a choice: (CIRCLE O).

2.2 Spatial Relation Predicates

Combined with low level glyph recognition, a set of binary spatial relations -- predicates on glyphs -- are building blocks for higher-level recognition and interpretation. The relations (such as 'contains,' 'immediately-above,' 'lines-tee-connect,' 'same-size,' 'overlaps') examine coordinate data in the raw glyphs, usually just the bounding boxes, sometimes also first and last points. The program uses the predicates to produce symbolic descriptions of the elements and relations in a diagram (figure 6), and to recognize arrangements of simple glyphs.



```
(CONTAINS BOX CIRCLE)
(CONTAINS BOX TRIANGLE)
(IMMEDIATELY-ABOVE BOX TOP)
(IMMEDIATELY-RIGHT-OF CIRCLE TRIANGLE)
(SAME-SIZE CIRCLE TRIANGLE)
```

Figure 6: The program produces an analysis of spatial relations among diagram glyphs.

The spatial predicates are organized according to various metrics, for example by argument types: shapes such as circles and boxes overlap, but lines intersect. Some are also ordered by specificity: concentric is more specific than contains, in turn more specific than overlaps (see figure 9). Some relations are exclusive: two glyphs can be either adjacent, near, or far. These organizations of the predicates help make search more efficient and eliminate redundant descriptions; if two circles are concentric, the program need not note that one also contains the other. The program can analyze all relations among all diagram glyphs, a selected subset of relations, or a subset of glyphs. For example, we can ask to view "only adjacency relations," or "all relations involving boxes and arrows."

2.3 Higher-Level Recognition

The program uses a bottom-up parsing approach to assemble simple glyphs into higher-level configurations. After the low-level recognizer has tried to identify the glyphs drawn on the tablet, the higher-level recognizers come into play. Higher-level recognition is essentially a process of graphical search and replace [13, 14]. Each higher-level recognizer matches a specific configuration of glyphs arranged in certain spatial relations. For example, a "tree-diagrams" recognizer looks for circles, one above the other, linked by line segments. A "rooms" recognizer looks for words in boxes. A "poly-lines" recognizer looks for lines that connect or tee with another line. (Figure 7).

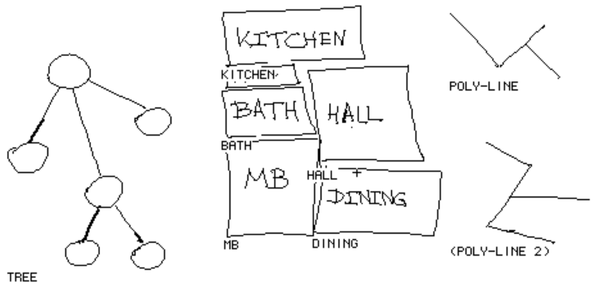


Figure 7: Each higher-level recognizer matches a specific configuration of glyphs arranged in certain spatial relations. For example, recognizers have been built for tree diagrams, rooms, and poly-lines.

The higher-level recognizers can be invoked explicitly by the designer ("identify the rooms in the diagram"), or they can be slated to run automatically whenever the program is quiescent, when the designer pauses after drawing. When a higher-level recognizer identifies its target configuration -- a word, a poly-line, a tree-diagram -- it makes a new compound glyph that stores the original low-level glyphs as parts. The parts can be left in the diagram for other recognizers to consider as well, or they can be removed from the diagram by the first recognizer to use them. The higher-level recognizers solve a constraint satisfaction problem over the diagram glyphs [10, 14] Each higher level recognizer finds instances of glyph sets $\{g_1, g_2, \ldots g_n\}$ that satisfy a set of relations $r_{i,j,k}$ where g_i and g_j are glyphs and $r_{i,j,[1\ldots k]}$ are binary relations between them. For example, the 'tree-diagram' recognizer collects glyph triples (g_1, g_2, g_3) such that the following conjunction of constraints holds:

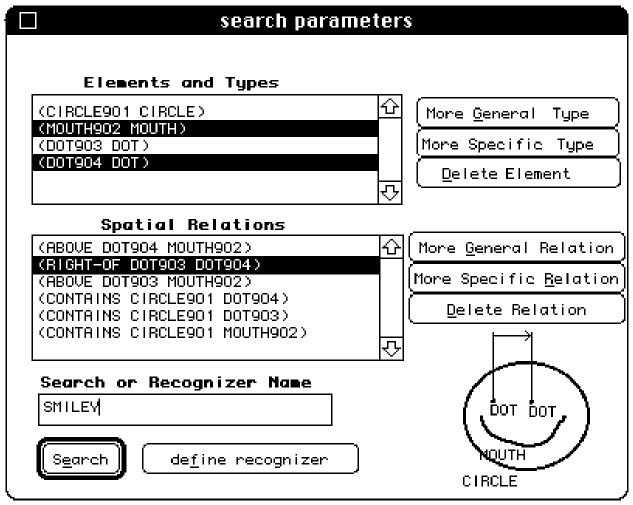
```
(TYPE G1 CIRCLE) (TYPE G2 CIRCLE) (TYPE G3 LINE)
(CONNECTS G3 G1) (CONNECTS G3 G2)
(ABOVE G1 G2)
```

In other words, the tree recognizer collects sets of three glyphs such that the first two are circles and the third is a line; the line connects the two circles, and one circle is above the other. The next version of the program will use constraint satisfaction for higher-level recognition, based on an implementation of v. Hentenryck et al.'s arc-consistency framework [11]. However, because the number of glyphs in a diagram is small, exhaustive search has been surprisingly acceptable for constructing simple higher-level recognizers.

Higher-level recognizers can resolve ambiguities that the low-level recognizer cannot. For example, the program has been trained to identify one glyph as both "Circle" and "Letter O." Therefore the low-level recognizer, working one glyph at a time, should always produce a multiple choice. Later the higher-level word recognizer can determine if the circle/letter-O glyph is next to another letter of approximately the same size. Then the glyph becomes a "Letter-O". If instead the same glyph appears in a tree then the tree-diagram recognizer declares it to be a "Circle".

2.4 Constructing higher-level recognizers

To specify a search pattern or to construct a new higher-level recognizer a user begins by selecting a set of elements in the diagram. A dialog displays a textual description of the elements, their types, and the observed spatial relations, which are annotated graphically on the diagram elements as well (figure 8). The dialog is used to control graphical search and to define higher-level recognizers.


search parameters

Elements and Types

```
(CIRCLE901 CIRCLE)
(MOUTH902 MOUTH)
(DOT903 DOT)
(DOT904 DOT)
```

Spatial Relations

```
(ABOVE DOT904 MOUTH902)
(RIGHT-OF DOT903 DOT904)
(ABOVE DOT903 MOUTH902)
(CONTAINS CIRCLE901 DOT904)
(CONTAINS CIRCLE901 DOT903)
(CONTAINS CIRCLE901 MOUTH902)
```

Search or Recognizer Name: SMILEY

Figure 8: The search dialog provides an analysis of glyph types and relations that the user can adjust, making descriptions more or less specific, and deleting unwanted parts of the description.

The user then refines the search pattern by deleting elements or relations not intended as part of the search pattern and by making element types and spatial relations more or less specific. Figure 9 shows fragments of the "specific-general" graph that orders spatial relations.

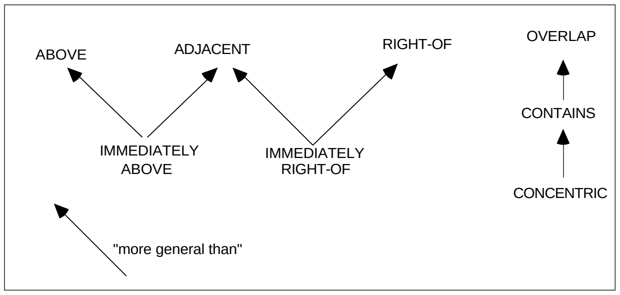


Figure 9: "More-general" relation orders spatial predicates. Adjustments made in the search dialog traverse this graph, to tighten or loosen descriptions of glyph configurations.

For example, if a user selects as a search pattern a box with a letter "A" immediately to its right then the program generates the search pattern of figure 10a, matching any letter that is immediately to the right of a box.

```
(TYPE G1 BOX)                      (TYPE G1 SHAPE)
(TYPE G2 LETTER)                   (TYPE G2 LETTER-A)
(IMMEDIATELY-RIGHT-OF G1 G2)       (ADJACENT G1 G2)
```

Figure 10 (a, b): The general and specific buttons in the search dialog are used to adjust a search pattern. Pattern at left matches any letter immediately to right of a box. Pattern at right matches a letter-A adjacent to any shape.

Using the "more general" and "more specific" commands, the user modifies the search pattern to the one in figure 10b, which matches only the letter "A", adjacent to (immediately above, below, left, or right of) any shape. The scope of search is limited to two options: the current diagram, or the current diagram plus all diagrams in the catalog. Thus, a search could retrieve (for example) all floorplan variants with the kitchen adjacent to the entry or every page with a phone-number written on it.

Once the user has defined a search pattern, it can be saved and reused for subsequent searches. It can be used to define as a higher-level recognizer, which replaces occurrences of the search configuration with a new, compound object, and added to the list of recognizers that operate constantly in the background. Figure 8 shows the definition of a higher-level-recognizer for “smileys”.

3. DIAGRAM MANAGEMENT

Apart from recognition and interpretation, but equally essential in design, is the drawing environment and the management of diagrams. The cocktail napkin prototype supports selective rectification, gestural commands, simulated tracing paper, an on-screen catalog of previously made diagrams, and a simple two-person shared drawing space. Although these are all simple features, they are the elements of an environment for designing.

3.1 Selective Rectification

Although once the program has recognized a glyph it can replace the hand-drawn version with a "rectified" symbol (e.g. a straight-line, 90° rectangle replaces a crude box), designers may well prefer to retain the sloppy -- and perhaps suggestive -- hand drawn version at least until they are further along in designing. Therefore rectification is performed on a per-glyph basis and the hand-drawn input data is always retained. The program can rectify all new glyphs or leave them as drawn and the designer can always change the display of particular glyphs.

3.2 Gestural Commands

The designer controls the program using a short menu of command buttons (left, figure 1). Frequently used commands such as erase and undo can also be executed by gestures, which are trained and recognized just as any other glyph. A gestural command can operate on all the glyphs it touches or on the currently selected glyph or glyphs. The designer can touch a glyph to select it or circle several glyphs and pick the circle to select its contents. Figure 11 shows the default glyphs for several common commands. Users can redefine the gesture for any commands simply by training a new glyph. To help users learn the gestural versions of commands, the program has a “novice mode” that displays the command gesture whenever the user invokes the menu-button version of the command.

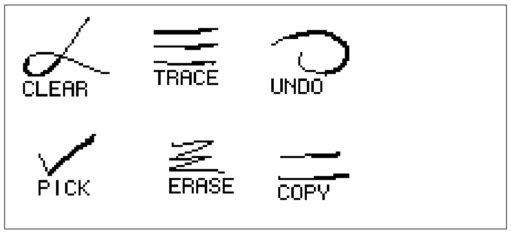


Figure 11. A simple set of gestural commands is provided. Users can change the default command glyphs.

3.3 Trace and Transparency

Tracing paper plays an important role in architectural and engineering design. For example, a designer can use it to copy most of a diagram from an underlay, making changes to certain parts, working all the time in the visual context of the original design. At the risk of mixing metaphors, the cocktail napkin prototype therefore includes a simulation of tracing paper's essential functions. The "trace" command grays the drawing window and lightens the glyphs to simulate overlaying a sheet of tracing paper. The display routines use graphical transparency to indicate which layer a glyph is on -- the lower the layer, the lighter the glyph. The program keeps track of multiple layers; tabs displayed on the upper left of the drawing area (see figure 1) allow a designer to select, remove, and replace trace layers. Glyphs on lower layers may be copied to the top layer, but only glyphs on the top layer may be erased.

3.4 What to do with all those napkins?

Although the occasional diagram is detailed and complicated, typically in early design many simple diagrams are made quickly and set aside. The cocktail napkin provides an on-screen catalog (see figure 1) for storing and viewing old diagrams, reduced to the size of icons. These old diagrams can be selected and recalled to full size, either replacing the current diagram, or brought in as a layer of trace. An obvious next step is to link related diagrams, for example, diagrams traced, or combined from others.

3.5 Inferring Context

Glyphs come in categories - the alphabet, simple shapes, circuit symbols, floorplan elements. The training sets for these symbols are stored in separate files, though they are trained together to minimize interference. Each diagram stores a list of glyph categories that it contains, which the program can use to resolve ambiguity. For example, a looped squiggle might be a spring or a coil; if the diagram already has other electrical symbols but no mechanical ones, then the program identifies the symbol as a coil. Of course the user can override this identification.

3.6 Shared Drawing Surface

The program also supports a simple form of collaborative diagramming, where designers draw on the same tablet using different pens or on separate tablets. Both designers' marks appear on the same diagram in different colors, so it is easy to see who made what mark.

The two pen version uses two different digitizing pens that produce different pressure values, and the software samples these values to determine which pen is being used. (WACOM Technology Corp. kindly provided several pens to experiment with. Simplest is to use a standard pen and a pressure-sensing pen.) Since the pressure-value profiles of the pens may overlap, the tablet input routine samples pressure values of the first few data points just before or just after the pen hits the tablet to determine which pen is drawing. The tablet hardware allows only one pen to draw at a time but that is also true of small cocktail napkins.

The two tablet version uses both Macintosh serial ports. Although the designers use separate physical drawing surfaces share one work area. The tablet driver arbitrates between ports and serves whoever first starts drawing, locking out the other user. When one user finishes a glyph the other can begin to draw.

4. INITIAL EXPERIENCE WITH USERS

The cocktail napkin prototype is not stable and robust enough for extensive and formal user testing but throughout development we have subjected a dozen novice users to its evolving interface (mostly undergraduate architecture students enrolled in a senior seminar on “the future of CAD.”) Although often frustrating, this form of testing-in-development has been tremendously valuable in identifying poorly designed features and interface bugs at a time when they are still easy to fix. In several instances the sometimes naive suggestions of user-testers have resulted in immediate improvements to the interface.

There is still a considerable learning period (2-4 hours) before users can obtain reasonable recognition rates from the program. To be sure, the low-level recognition is imperfect and its training set is individualized to a single way of drawing the glyphs. But a good part of the time is devoted to learning the feel of the tablet-pen ensemble, the pressure sensitivity range of the pen, and adjusting hand-eye coordination to the unfamiliar act of drawing on the tablet while looking at the screen. We observed shorter learning periods when we put paper on the tablet and ink in the pen and encouraged users to ignore the computer monitor and look only at the paper. This suggests that using a touch sensitive display (as in notepad computers) would shorten the learning period.

Once they become familiar enough with the hardware to obtain satisfactory recognition rates, users seem to have little difficulty performing the basic operations of the cocktail napkin interface. After a few hours of practice with the program novice users seem to enjoy executing gestural commands, adding new glyphs to the training set, using the simulated tracing paper, and even defining search patterns and higher-level recognizers.

5. CONCLUSIONS

The recognition of simple glyphs in diagrams is tractable in part because the universe of symbols is small. Even a simple recognition algorithm for low-level glyphs works surprisingly well Because the user can train the recognizer the program can accommodate new symbols and idiosyncratic ways of drawing. The cocktail napkin prototype handles search and recognition of higher-level arrangements of diagram symbols; a simple interface for programming this search takes advantage of ordering elements and relations in a "more general and specific" hierarchy. In addition to recognition and interpretation, the management of diagrams is equally essential for design. The simulated tracing paper, and the searchable catalog are our first efforts in this direction.

We’re considering a variety of “next steps.” For example, it will be relatively straightforward to connect the cocktail napkin with a small case-library of floor plans, indexed on room adjacencies. Then the program can provide case-based examples and advice in response to a user’s sketched bubble-diagram floor plan. Another direction is to strengthen the interface and routines for building higher-level recognizers from examples. For example, we have ignored cardinality -- a simple diagram of a “sun” (a circle with radial lines emanating from it) may have five, six, or seven lines. Should the recognizer automatically generalize cardinality in the search pattern?

We have in mind a larger diagram-based environment for design, one in which the program generates [4], as well as interprets diagrams, and one that is connected to a variety of tools for symbolic processing. We expect our emphasis will shift away from recognition toward the use of diagrams in design reasoning, and how computers can support this process.

ACKNOWLEDGMENTS

My area of interest is making design tools that serve designers better, by both a careful understanding of designing and appropriate use of computational techniques. My previous work has been largely in the application of constraint based techniques to design; the cocktail napkin project is a first effort in supporting diagramming. Since this paper was written we have used the program to build a query-by-diagram retrieval scheme for visual databases and for a constraint based diagrammer, Stretch-a-Sketch. Reports on these projects are available from the author.

Jim Davis at the Cornell Design Research Institute, Aaron Fleisher at MIT, and Kyle Kuczun in our group provided valuable arguments. The AVI reviewers' comments were also extremely helpful. Funds are being sought to continue the cocktail napkin project, which was pursued as an unfunded pilot project. The construction of constraint management routines (developed for a different project) was supported by National Science Foundation grant DMII 93-13186.

REFERENCES

1. Apte, A., Vo, V., Kimura, T.D. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In: Proc. ACM conference on User Interface and Software Technology (UIST). (Atlanta, GA, 1993), ACM, pp. 121-128.

2. Bly, S., Harrison, S., Irwin, S. Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment. Communications of the ACM, 36,1, (1993) pp. 26-45.

3. Chandrasekaran, B., Narayanan, N.H., Iwasaki, Y. Reasoning with Diagrammatic Representations. AI Magazine, 14,2, (1993) pp. 49-56.

4. Ervin, S.M. Designing with Diagrams. In: The Electronic Design Studio: Architectural Knowledge and Media in the Computer Age. McCullough M, Mitchell WJ, Purcell P, Ed. MIT Press, Cambridge, MA, 1990, pp. 107-122.

5. Futrelle, R., Kakadiaris, I.A., Alexander, J., Carriero, C.M., Nikolakis, N., Futrelle, J.M. Understanding Diagrams in Technical Documents. IEEE Computer, July,(1992) pp. 75-78.

6. Gero, J.S., ed. Artificial Intelligence in Design '92. Oxford, UK: Butterworth-Heinemann, 1992.

7. Goel, V. "Ill-structured representations" for ill-structured problems. In: Proc. Fourteenth Annual Conference of the Cognitive Science Society. (Bloomington, IN, 1992), Hillsdale, NJ: Erlbaum, pp. 844-849.

8. Goldschmidt, G. The dialectics of sketching. Creativity Research Journal, 4,2, (1991) pp. 122-143.

9. Golin, E., Reiss, S.P. The specification of visual language syntax. In: Proc. IEEE Workshop on Visual Languages. (1989), IEEE Press, pp. 105-110.

10. Helm, R., Marriott, K., Odersky, M. Building Visual Language Parsers. In: Proc. Human Factors in Computing Systems (CHI '91). (New Orleans, LA, 1991), ACM Press / Addison Wesley, pp. 105-112.

11. Hentenryck, P.V., Deville, Y., Teng, C.-M. A generic arc-consistency algorithm and its specializations. Artificial Intelligence, 57, (1992) pp. 291-321.

12. Ishii, H., Miyake, N. Toward an Open Shared Workspace: Computer and Video Fusion Approach of Team Workstation. Communications of the ACM, 34,12, (1991) pp. 37-50.

13. Kurlander, D., Bier, E. Graphical Search and Replace. In: Proc. SIGGRAPH. (Atlanta, GA, 1988), ACM Press, pp. 113-120.

14. Kurlander, D., Feiner, S. Interactive Constraint Based Search and Replace. In: Proc.

Human Factors and Computing Systems (CHI ‘92). 1992, pp. 609-618.

15. Lakin, F., Wambaugh, J., Leifer, L., Cannon, D., Steward, C. The electronic notebook: performing medium and processing medium. Visual Computer, 5,(1989) pp. 214-226.

16. Larkin, J., Simon, H. Why a diagram is (sometimes) worth 10,000 words. Cognitive Science, 11,(1987) pp. 65-99.

17. Negroponte, N. Recent advances in sketch recognition. In: Proc. AFIPS (American Federation of Information Processing) National Computer Conference. (Boston, MA, 1973), pp. 663-675.

18. Rubine, D. Specifying Gestures by Example. Computer Graphics, 25,4, (1991) pp. 329-337.

19. van Sommers, P. *Drawing and Cognition - Descriptive and experimental studies of graphic production processes*. Cambridge University Press, Cambridge, England, 1984.

20. Wittenburg, K., Weitzman, L. Visual Grammars and Incremental Parsing. In: Proc. IEEE Workshop on Visual Languages. (Skokie, IL, 1990), pp. 235-243.

21. Zhao, R. Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors. In: Proc. INTERCHI ‘93. (Amsterdam, 1993), ACM / Addison Wesley, pp. 95-100.