Sketch Understanding in Design: Overview of Work at the MIT AI Lab

Randall Davis
MIT Artificial Intelligence Laboratory
davis@ai.mit.edu

Abstract

We have been working on a variety of projects aimed at providing natural forms of interaction with computers, centered primarily around the use of sketch understanding. We argue that sketch understanding is a knowledge-based task, i.e., one that requires various degrees of understanding of the act of sketching, of the domain, and of the task being supported. In the long term we aim to use sketching as part of a design environment in which design rationale capture is a natural and, ideally, almost effortless byproduct of design.

Natural Interaction

We suggest that the problem with software is not that it needs a good user interface, but that it needs to have *no* user interface. Interacting with software should – ideally – feel as natural, informal, rich, and easy as working with a human assistant.

As a motivating example, consider a hand-drawn sketch of a design for a circuit breaker (Fig. 1):

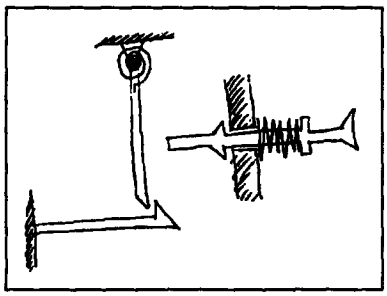


Fig. 1: Sketch of a circuit breaker design.

A typical spoken explanation of its intended behavior would indicate:

> *The current flows into the lever [pointing to wire at the top of the sketch], down to the hook, and out here [pointing to left of hook]. This [pointing to hook] is a bimetallic strip; when enough current flows it heats and bends down, allowing the lever to rotate [gesturing counter-clockwise] under the force of the coil spring.*

When given this explanation and asked "Do you understand how this works?" most people say "yes." This raises a number of interesting points. First, what do they mean by saying that they *understand*? One aspect of understanding is the ability to "run a movie in one's head," i.e., a mental simulation that sees the device in operation and can make predictions about its behavior. Another aspect is the ability to infer the intended function of components not explicitly described. What, for example, is the function of the components on the right side of the sketch? Engineers (mechanical or otherwise) see almost immediately that it is a reset button.

Our long term goal is to enable computers to do just what people do when presented with these sorts of sketches and explanations: We want to be able to draw a sketch like that in Fig. 1, say aloud the same 42 words, and make the same gestures, and have the computer reply that it understands, meaning by that the same thing we do.

While this is clearly a tall order, it is also one crucial step toward a much more natural style of interaction with computers. The work in our group is aimed at doing this, making it possible for people involved in design and planning tasks to sketch, gesture, and talk about their ideas (rather than type, point, and click), and have the computer understand their messy freehand sketches, their casual gestures, and the fragmentary utterances that are part and parcel of such interaction.

One key to this lies in appropriate use of each of the means of interaction: Geometry is best sketched, behavior and rationale are best described in words and gestures.

A second key lies in the claim that interaction will be effortless only if the listener is smart: effortless interaction and invisible interfaces must be knowledge-based. If it is to make sense of informal sketches, the listener has to understand something about the domain and something about how sketches are drawn.

This paper provides an overview of nine current pieces of work at the MIT AI Lab in the Design Rationale Capture group on the sketch recognition part of this overall goal.

Early Processing

The focus in this part of our work is on the first step in sketch understanding: interpreting the pixels produced by the user's strokes, producing low level geometric descriptions such as lines, ovals, rectangles, arbitrary polylines, curves and their combinations. Conversion from pixels to geometric objects provides a more compact

representation and sets the stage for further, more abstract interpretation.

Our initial domain – mechanical engineering design – presents the interesting (and apparently common) difficulty that there is no fixed set of shapes to be recognized. While there are a number of traditional symbols with somewhat predictable geometries (e.g., symbols for springs, pin joints, etc.), the system must also be able to deal with bodies with arbitrary shapes composed of both straight lines and curves. As consequence, accurate early processing of the basic geometry – finding corners, fitting both lines and curves – becomes particularly important.

Our approach takes advantage of the interactive nature of sketching, combining information from both stroke direction and stroke speed data. Consider as an example the square in Fig 2 along with curves showing the direction and speed data for this stroke.

The general idea is to locate vertices by looking for points along the stroke that are minima of speed (the pen slows at corners) or maxima of the absolute value of curvature. But noise in the data introduces many false positives, while false negatives result from subtle changes in speed or curvature (e.g., in polylines formed from a combination of very short and long line segments, the maximum speed reached along the short line segments may not be high enough to indicate the pen has started traversing another edge, with the result that the entire short segment is interpreted as the corner). This problem arises frequently when drawing thin rectangles, common in mechanical devices.

To deal with these difficulties we use average based filtering, and a technique that combines information from both speed and curvature. Average based filtering looks for extrema only in areas of the speed and curvature data that exceed the average value (see [Sezgin01] for details). This reduces (but does not eliminate) false positives.

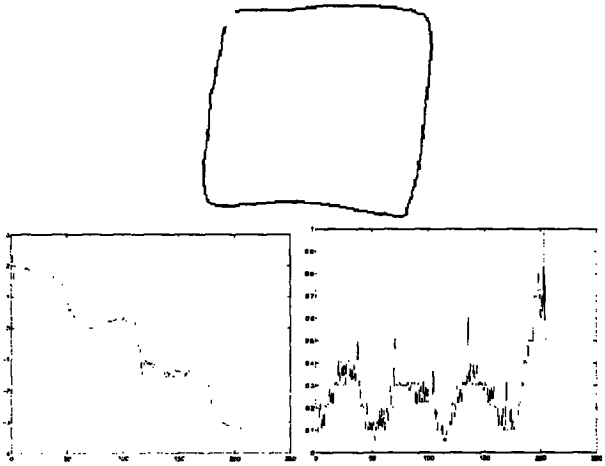


Fig 2: A hand-drawn square, point-to-point direction, and point-to-point speed.

We then combine both sources of information, generating hybrid fits by combining the set of candidate vertices derived from (average-filtered) curvature data with the candidate set from filtered speed data, taking into account the system's certainty that each candidate is a real vertex. Points where both sources of evidence suggest a vertex are the strongest candidates; additional candidates are selected from the most points most strongly supported by either speed or direction data alone (see [Sezgin01]for details).

The polyline approximation generated by this process provides a natural foundation for detecting areas of curvature: we compare the Euclidean distance between each pair of consecutive vertices in our fit from above, to the accumulated arc length between those vertices in the input. The ratio of these is very close to 1 in linear regions of the input and significantly higher than 1 in curved regions. We approximate curved regions with Bezier curves.

Two examples of the capability of our approach is shown below, in a pair of hand-sketched mixture of lines and curves. Note that all of the curved segments have been modeled with curves, rather than the piecewise linear approximations that have been widely used previously.

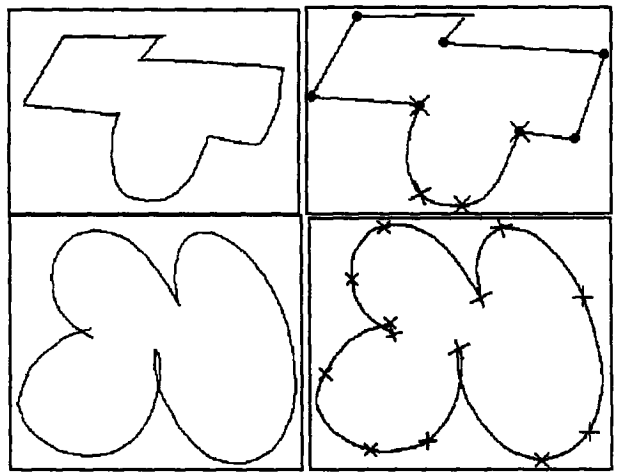


Fig 3: Input sketch at left; analyzed strokes at right (dots indicate detected vertices, x's indicate beginning and end of detected curved segments).

We have conducted a user study to measure the degree to which the system is perceived as easy to use, natural and efficient. Study participants were asked to create a set of shapes using our system and Xfig, a Unix tool for creating diagrams. Xfig is a useful point of comparison because it is representative of the kinds of tools that are available for drawing diagrams using explicit indication of shape (i.e., the user indicates explicitly which parts of the sketch are supposed to be straight lines, which curves, etc.)

Overall, users praised our system because it let them draw shapes containing curves and lines directly and without having to switch back and forth between tools. We have also observed that with our system, users found it much easier to draw shapes corresponding to the gestures they routinely draw freehand, such as a star.

Device Recognition

One important step toward sketch understanding is resolving ambiguities in the sketch–determining, for example, whether a circle is intended to indicate a wheel or a pin joint–and doing this as the user draws, so that it doesn't interfere with the design process. We have developed a method and an implemented program that does this for freehand sketches of simple 2-D mechanical devices.

Our work in this part is focused on creating a framework in which to represent and use contextual (top-down) knowledge to resolve ambiguities. We built a program called ASSIST (A Shrewd Sketch Interpretation and Simulation Tool) that interprets and understands a user's sketch as it is being drawn, providing a natural-feeling environment for mechanical engineering sketches [Alvarado01a].

The program has a number of interesting capabilities:

- Sketch interpretation happens in real time, as the sketch is being created.
- The program allows the user to draw mechanical components just as on paper, i.e., as informal sketches, without having to pre-select icons or explicitly identify the components.
- The program uses a general architecture for both representing ambiguities and adding contextual knowledge to resolve the ambiguities.
- The program employs a variety of knowledge sources to resolve ambiguity, including knowledge of drawing style and of mechanical engineering design.
- The program understands the sketch, in the sense that it recognizes patterns of strokes as depicting particular components, and illustrates its understanding by running a simulation of the device, giving designers a way to simulate their designs as they sketch them.

Fig 4a shows a session in which the user has drawn a simple car on a hill. The user might begin by drawing the body of the car, a free-form closed polygon. As the user completes the polygon, the system displays its interpretation by replacing the hand-drawn lines with straight blue lines. Next the user might add the wheels of the car, which also turn blue as they are recognized as circular bodies. The user can then "attach" the wheels with pin joints that connect wheels to the car body and allow them to rotate. The user might then draw a surface for the car to roll down, and anchor it to the background (the "x" indicates anchoring; anything not anchored can fall). Finally, the user can add gravity by drawing a downward pointing arrow not attached to any object. The user's drawing as re-displayed by ASSIST is shown in Fig 4b. When the "Run" button is tapped, it transfers the design to a two-dimensional mechanical simulator which shows what will happen (Fig 4c).

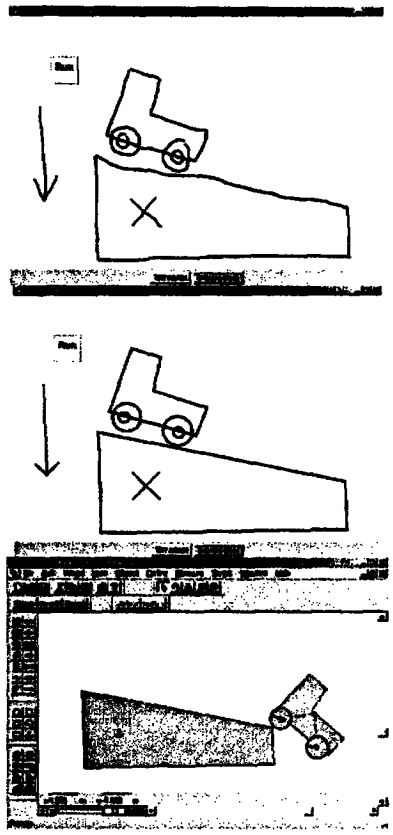


Fig 4a, b, c: A session with ASSIST.

Note that the user drew the device without using icons, menu commands, or other means of pre-specifying the components being drawn. Note, too, that there are ambiguities in the sketch, e.g., both the wheels of the car and pin joints are drawn using circles, yet the system was able to select the correct interpretation, by using the knowledge and techniques discussed below. The automatic disambiguation allowed the user to sketch without interruption.

Note that ASSIST deals only with recognizing the mechanical components in the drawing and is, purposely, literal-minded in doing so. Components are assembled just as the user drew them, and component parameters (e.g. spring constants, magnitudes of forces, etc.) are set to default values. The car above, for example, wobbles as it runs down the hill because the axles were not drawn in the center of the wheels. The combination of literal-minded interpretation and default parameter values can produce device behavior other than what the user had in mind. Other work in our group, discussed below, has explored the interesting and difficult problem of communicating and understanding the *intended* behavior of a device .

ASSIST'S overall control structure is a hierarchical template-matching process, implemented in a way that produces continual, incremental interpretation and re-evaluation as each new stroke is added to the sketch. Each new stroke triggers a three stage process of recognition, reasoning and resolution. Recognition generates all

possible interpretations of the sketch in its current state, reasoning scores each interpretation, and resolution selects the current best consistent interpretation. After each pass through the three stages the system displays its current best interpretation by redrawing the sketch.

In the recognition stage, ASSIST uses a body of recognizers, small routines that parse the sketch, accumulating all possible interpretations as the user draws each stroke.

In the reasoning stage the system scores each interpretation using several different sources of knowledge that embody heuristics about how people draw and how mechanical parts combine. Those sources include:

Temporal Evidence: People tend to draw all of one object before moving to a new one. Our system considers interpretations that were drawn with consecutive strokes to be more likely than those drawn with non-consecutive strokes.

Simpler Is Better: We apply Occam's razor and prefer to fit the fewest parts possible to a given set of strokes.

Domain Knowledge: ASSIST uses basic knowledge about how mechanical components combine. For example, a small circle drawn on top of a body is more likely to be a pin joint than a circular body.

User Feedback: User feedback also supplies guidance. A "Try Again" button permits the user to indicate that something was recognized incorrectly, at which point the system discards that interpretation and offers the user an ordered list of alternative interpretations. Conversely the system can be relatively sure an interpretation is correct if the user implicitly accepts it by continuing to draw.

The heuristics described above all independently provide evidence concerning which interpretation is likely to be correct. Our method of combining these independent sources involves distinguishing between two categories of evidence: categorical and situational, and is described in detail in [Avarado01a].

The third stage in the interpretation process involves deciding which interpretation is currently the most likely. Our system uses a greedy algorithm, choosing the interpretation with the highest total score, eliminating all interpretations inconsistent with that choice, and repeating these two steps until no more interpretations remain to be selected. Details of all three phases are in [Alvarado01a].

Our initial evaluation of ASSIST has focused on its naturalness and effectiveness. We asked subjects to sketch both on paper and using ASSIST. We observed their behavior and asked them to describe how ASSIST felt natural and what was awkward about using it. All were asked first to draw a number of devices on paper, to give them a point of comparison and to allow use to observe differences in using the two media.

The system was successful at interpreting the drawings despite substantial degrees of ambiguity, largely eliminating the need for the user to specify what he was drawing. As a consequence, a user's drawing style appeared to be only mildly more constrained than when drawing on paper.

People reported that the system usually got the correct interpretation of their sketch. Where the system did err, examination of its performance indicated that in many cases the correct interpretation had never been generated at the recognition step, suggesting that our reasoning heuristics are sound, but we must improve the low-level recognizers. This work is currently under way.

Users tended to draw more slowly and more precisely with ASSIST than they did on paper. The most common complaint was that it was difficult to do an accurate drawing because the system changed the input strokes slightly when it re-drew them (to indicate its interpretations). Users felt that the feedback given by ASSIST was effective but at times intrusive. Our next generation of the system leaves the path of the strokes unchanged, changing only their color to indicate the interpretation.

For a more complete discussion responses to the system from a user interface perspective, see [Alvarado01b].

Conveying Intended Behavior

So far we have the ability to recognize components and how they are connected. But the intended behavior of a device is not always obvious from its structure alone. Consider the (whimsical) egg-cracking device shown below (adapted from [Narayanan95]):

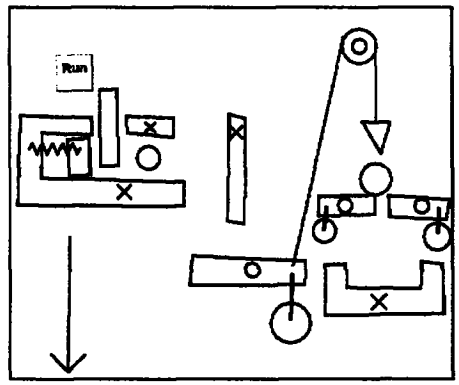


Figure 5: Sketch of a whimsical device.

The intent is that, as the stopper (the vertical bar near the run button) is pulled up, the spring forces the ball to the right, it falls onto the see-saw, allowing the wedge to drop, cracking the egg into the frying pan.

But if we simply run the simulation, nothing interesting happens: the stopper, responding to gravity, simply drops down a little, as does the ball, which then stays put. We need to be able to tell the system exactly the information in the paragraph under Figure 5, and have it understand.

Designers routinely do this, explaining their designs to one another using sketches and verbal explanations of behavior, both of which can be understood long before the device has been fully specified. But current design tools fail almost completely to support this sort of interaction, instead forcing designers to specify details of the design by navigating a forest of menus and dialog boxes, rather than

directly describing the behaviors with sketches and verbal explanations. We have created a prototype system, called ASSISTANCE, capable of interpreting multi-modal explanations for simple 2-D kinematic devices [Oltmans01].

The program generates a model of the events and the causal relationships between events that have been described via hand drawn sketches, sketched annotations, and verbal descriptions. Our goal is to make the designer's interaction with the computer more like interacting with another designer. This requires the ability not only to understand physical devices but also to understand the means by which the explanations of these devices are conveyed.

As a trivial yet instructive example, consider a spring attached to a block positioned next to a ball. In a traditional CAD system the designer would select the components from a tool bar and position them, and would then have to specify a variety of parameters, such as the rest length of the spring, the spring constant, etc. (Fig 6a). Contrast this to the way someone would describe this device to a colleague. As we discovered in a set of informal experiments, the description typically consists of a quick hand drawn sketch and a brief spoken description, "the block pushes the ball." In response, we have built a tool that augments structural descriptions by understanding graphical and verbal descriptions of behavior.

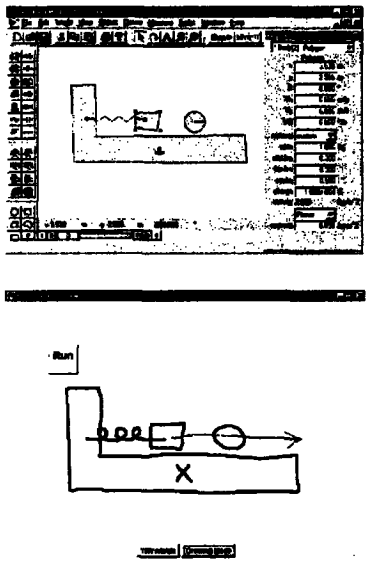


Fig 6: A block and ball described in a CAD-style tool, and as a sketch.

ASSISTANCE can currently understand descriptions of two dimensional kinematic devices that use rigid bodies, pin joints, pulleys, rods, and springs. It takes spoken natural language and hand-drawn sketches as input and generates a causal model that describes the actions the device performs and the causal connections between them.

We take "understanding" in this context to mean the ability to generate a causal model that accurately reflects the behavior description given by the designer. The system's task is thus to understand the designer, without attempting to determine whether the designer's description is physically accurate.

The representations ASSISTANCE generates are not a verbatim recording of the designer's description. To demonstrate that it has understood an explanation (and not just recorded it), ASSISTANCE can construct simple explanations about the role of each component in terms of the events that it is involved in and causal connections between events. Further evidence of the system's understanding is provided by its ability to infer from the behavior description what values some device parameters (e.g., spring constants) must take on in order to be consistent with the description. Because our current work has focused on building the model, the query and parameter adjustment capabilities are designed only to provide a mechanism for the system to describe its internal model and to suggest how such representations could be used in the future. We do not yet attempt to deal with the difficult issues of explanation generation, dialog management, or general parametric adjustments.

Our current implementation makes the task tractable by taking advantage of a number of sources of knowledge and focusing the scope of the task. Our focus on two-dimensional kinematic devices, limits the vocabulary and grammar necessary to describe a device, making the language understanding problem tractable. We then take advantage of two characteristics of informal behavior descriptions: they typically contain overlapping information and they are often expressed in stereotypical forms. We use the multiple, overlapping descriptions of an event--the same event described in a verbal explanation and in a sketched annotation--to help infer the meaning of the description. We also combine multiple descriptions to produce a richer description than either one provides alone. Finally, we use knowledge about the way designers describe devices to simplify the process of interpreting their descriptions (e.g., mechanical device behavior is frequently described in the order in which it occurs).

ASSISTANCE begins with a description of the device's structure that specifies each of the objects in the figure and their connections, and does a degree of freedom analysis based on the interconnection information (e.g., anchors prevent both rotation and translation while pin joints allow rotation).

The bulk of the work of ASSISTANCE lies in parsing the user's verbal description and sketched annotations, and providing a causal model of the device behavior. We walk through one input to illustrate this process in action, detailing the knowledge required to understand the description. The example illustrates assistance's ability to infer motions of bodies, identify multiple descriptions of the same event, disambiguate deictic references, and infer causal links between motions.

When the says "When the stopper moves up the spring releases." ASSISTANCE begins by breaking the utterance into its constituent clauses and translates them into events.

A straightforward interpretation of the first clause ("The stopper moves up") generates a representation for the motion of that body. The system then infers the motion of the piston from the second clause ("the spring releases"), based on the observation that the spring is connected on the left end to an anchored body, hence in order for the spring to "release," the piston must be moving. This is an example of an inference based on the physical structure of the device.

ASSISTANCE then infers a causal connection between these two motions because the two clauses are linked by a conditional statement ("When the stopper moves...") suggesting causality, in which the motion of the first clause is a precondition for the motion in the second. This is an example of using linguistic properties to infer a causal link between events.

Speech recognition is handled by IBM's ViaVoice software, which parses the utterances against a grammar containing phrases we found commonly used in device descriptions. The grammar abstracts from the surface level syntactic features to an intermediate syntactic representation that explicitly encodes grammatical relations such as subject and object. These intermediate representations are used by rules (described below) to generate semantic representations of the utterances. This type of intermediate syntactic representation is similar to the approach taken in [Palmer93].

The grammar is written using the Java Speech Grammar Format, which provides a mechanism for annotating the grammar rules with tags. These tags decorate the parse tree generated by the speech recognition system with both the surface level syntactic features and the intermediate syntactic representations mentioned above.

The sketched gestures currently handled by ASSISTANCE are arrows and pointing gestures. Both of these gesture types are recognized by ASSIST and converted into a symbolic representation that includes the object that they refer to; ASSISTANCE then reasons with the symbolic representations. For arrows, the referent is the object closest to the base of the arrow and for pointing gestures it is the object that is closest to the point indicated.

After finding all the events and the causal relationships between them, ASSISTANCE has two remaining tasks: (i) find the set of consistent causal structures, and (ii) choose the causal structure that is closest to the designer's description.

Two constraints must be satisfied in order for a causal ordering to be considered consistent: (i) each event must have exactly one cause (but can have multiple effects), and (ii) causes precede effects.

The program tries all the plausible causes of each event until each has a cause. Any event that does not have a cause can be hypothesized to be caused by an exogenous force (a later step minimizes the number of hypothesized exogenous causes).

Finally, the system must choose from all the consistent models the one that most closely matches the designer's description. Two heuristics are used to select the model: there should be a minimal number of events caused by exogenous forces, and the order of the events in the causal description should be as close as possible to the order in which they were described (this heuristic is based on our empirical observation that people generally describe behavior in the order in which it occurs).

We have not yet performed a formal evaluation of assistance's naturalness but can offer comments from our own experiences. First, the process of representing the behavior of a device in ASSISTANCE is far more straightforward than interacting with a typical CAD program. The ability to describe behaviors independent of the parameters that lead to them is invaluable.

The primary difficulty currently is natural language processing. The grammar of recognized utterances is currently too small to allow designers who have not previously used the system to fluidly describe a device. This difficulty is complicated by occasional errors in the speech recognition. Future work needs to focus on ways in which the interface can subtly guide the user and let them know what types of utterances it will understand, without standing in the way of fluid explanations.

Building a New Architecture

As noted in [Alvarado02], we are working on a second generation of architecture for our sketch understander. We are designing a Hearsay-like architecture [Erman80], i.e., a multi-level blackboard populated by a collection of knowledge sources at a variety of levels of abstraction, all contributing asynchronously and independently to the interpretation. The lowest level knowledge sources will include the geometry recognizers that work with the raw strokes; component recognizers and behavior recognizers are at successively higher levels of detail, with the overall application at the highest level.

The blackboard framework has a number of advantages, including the ability to have knowledge sources make independent contributions to the interpretation. This in turn facilitates testing of the power and contributions of different modules, because they can easily be "swapped" in and out and the effect of their presence calibrated.

The framework also permits mixed top-down and bottom-up processing: knowledge sources can interpret existing data (bottom-up) or use the current interpretation as context to predict what ought to be present (top-down). The blackboard also facilitates working from "islands of certainty," i.e., starting at those places in the sketch where we are most certain of the interpretation and working outward from there. This can provide significant assistance dealing with ambiguity.

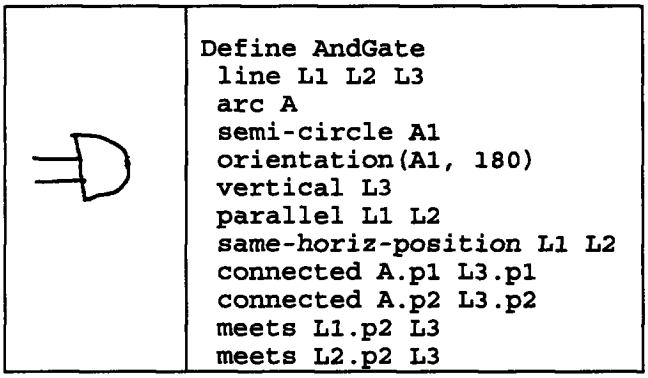
Perhaps most important, the Hearsay framework has proven to be an effective framework for organizing and deploying large bodies of knowledge (e.g., in speech understanding, acoustics, phonetics, syntax, semantics, and pragmatics). We believe that sketch understanding, no less than speech understanding, is a knowledge-intensive task.

Languages for Shape and Drawing Sequence

Building sketch recognizers (e.g., a spring recognizer, a pulley recognizer) is currently a process of analyzing a sketch by hand and writing code designed to look for what we believe to be the characteristic features of the object depicted. This is labor-intensive and the quality of the final code is too dependent on the style of the individual programmer.

We want the process to be far simpler, more principled and consistent. We have as a result begun to plan the development of a number of languages, including languages for describing shape, drawing, gestures, and behavior. The intent is that instead of simply writing code, a new shape recognizer will be added to the system's vocabulary by writing a description of the shape of the object, and providing an indication of how it is drawn (i.e., the sequence in which the strokes typically appear). A specialized compiler will take those descriptions and generate recognizer code from it.

We have a very early prototype language, developed by examining the symbols found in a variety of diagram languages, including mechanical designs, electronic circuit diagrams, and military symbology, but need to expand and extend the language and make it more robust. One example of the language is given below, for an and-gate:



```
Define AndGate
 line L1 L2 L3
 arc A
 semi-circle A1
 orientation(A1, 180)
 vertical L3
 parallel L1 L2
 same-horiz-position L1 L2
 connected A.p1 L3.p1
 connected A.p2 L3.p2
 meets L1.p2 L3
 meets L2.p2 L3
```

The next required element is a drawing sequence description language, i.e., a way to indicate how this symbol is typically drawn, so that we can take advantage of that knowledge when trying to recognize it. In this case, for example, the vertical bar of the gate is almost invariably drawn first, then the arc and finally the two wires.

While we could ask someone to write this down in a textual language like the one above, the far easier (and more obvious) thing to do is ask them to draw it a few times, have the system "watch" the sequence of strokes, then record that information in a drawing sequence description language we will create.

Learning New Icons

While writing a shape description of the sort shown above is far easier than writing the code for a recognizer, there is of course a still more natural way to describe a new shape to the system: draw it. Hence we are working toward a learning capability in which the user can draw a new icon once or twice, and the system would then generate the shape description and drawing sequence description.

Of these the shape description is far more difficult, as it requires abstracting from the specific image (with all of its noise and artifacts) just those properties that define the icon. In the hand-drawn and-gate above, for instance, every line has an exact length and orientation, yet it is the far more general properties of parallelism, equality of length, etc. that define the icon.

Our approach to generalization is based in part on data from the psychological literature that indicates what properties people naturally attend to. If shown the and-gate above, for instance, and then asked to describe it, people routinely attend a collection of fairly abstract relationships, ignoring much of the remaining detail [Goldmeier72]. We plan to use this to guide the descriptions produced by our system.

A Recognizer Generator

We are working to create a recognizer generator that would take descriptions of the sort shown in Table 1 and generate efficient code for recognizing that symbol. By efficient, we mean such things as taking account of spatial and temporal constraints: the individual strokes making up the icon will have been drawn in the same general place, and are likely to all have been drawn at roughly the same time. We believe that a recognizer that takes account of these and other constraints can be very efficient; the task here is to produce a generator smart enough to produce efficient code.

Multi-Modal Interaction

In one early demonstration of ASSIST a mechanical designer asked us to draw three identical, equally spaced pendulums. We were struck by how easy it was to say such a thing, and how difficult it was to draw it freehand. While standard editing commands (e.g., copy, move) might make the task easier, it is still far simpler, and importantly, far more natural to say such a thing than to have to do it graphically. We have also observed that people sketching a device frequently describe it as they do so, in informal fragments of language. This has led to our effort to enable multi-modal interaction, with careful attention to the appropriate use of each modality: sketching is clearly appropriate for communicating spatial information, while verbal descriptions easily specify other properties and relations. We are evaluating a number of speech understanding systems (e.g., ViaVoice and SpeechBuilder [Weinstein01]) and determining how to approach the frequently ungrammatical and fragmentary utterances encountered in this context.

An Electronic Drafting Table

Our drawing work to date has been done with whiteboard-based devices that use marker-sized and shaped

ultrasonic emitters, or with digitizing tablets. While these are usable, they do not feel as natural as using a pen or pencil on a flat surface. We are creating such an environment by developing an electronic drafting table fashioned from a sheet of plexiglas on which we have mounted a sheet of indium tin oxide (ITO), a transparent material with a resistance of 310 ohms/ft^2. Clamp connectors are used to ground the sheet in the middle of two opposite ends; sensors are attached to the four corners and connected to an analog to digital converter. The "pen" is a simple device that produces five volts at its tip.

Our current prototype uses an 8.5 x 11 in. sheet of ITO; pen positions are sampled at 300hz, with an impressive spatial resolution of 0.5mm. This should prove adequate to produce the feeling drawing with a fine-point pen.

The pen appears to the computer as a mouse; the strokes themselves will be produced by an LCD projector doing rear-projection onto the bottom surface of the table. This arrangement avoids the problems produced by other means of providing drawing environments, such as the shadows in a front-projection set up, and the unreliable signal capture from pen-sized ultrasonic emitters used on a table top (the signal is easily blocked by hands or arms resting on the table). The net result should be an environment that feels as natural as a traditional drawing surface, yet provides the advantages of an online medium.

Related Work

References to our work cited below contain detailed discussions of related work for the individual efforts. Overviews of comparable efforts at sketch understanding as a means of interaction are described in [Oviatt00], [Landay01], [Stahovich97] and [Forbus01].

Acknowledgements

The work reported here is supported by the MIT Oxygen Project and has been carried out by: Aaron Adler, Christine Alvarado, Tracy Hammond, Michael Oltmans, Metin Sezgin, and Olga Veselova.

References

[Alvarado01a] Alvarado, Christine and Davis, Randall (2001). Resolving ambiguities to create a natural sketch based interface. Proceedings of IJCAI-2001, August 2001.

[Alvarado01b] Alvarado, Christine and Davis, Randall (2001). Preserving the freedom of paper in a computer-based sketch tool. Proceedings of HCI International 2001.

[Alvarado02] Alvarado C, Oltmans M, Davis R, A Framework for Multi-Domain Sketch Recognition, proceedings of this symposium.

[Erman80] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser and D. Raj Reddy, The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty , *ACM Computing Surveys* Volume 12 , Issue 2 (1980) Pages 213-253

[Forbus01] Kenneth Forbus, R. Ferguson, and J. Usher. Towards a computational model of sketching. In *IUI '01*, 2001.

[Goldmeier72] Erich Goldmeier, "Similarity in Perceived Visual Forms", Psychological Issues Vol VIII, No.1, Monograph 29, International Universities Press, New York (1972).

[Landay01] James A. Landay and Brad A. Myers, "Sketching Interfaces: Toward More Human Interface Design." In IEEE Computer, 34(3), March 2001, pp. 56-64.

[Oltmans01] Oltmans M, and Davis, Randall (2001). Naturally Conveyed Explanations of Device Behavior. Proceedings of PUI-2001, November 2001.

[Oviatt00] Oviatt, S.L., Cohen, P.R., Wu, L.,Vergo, J., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J. & Ferro, D. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions, Human Computer Interaction, 2000, vol. 15, no. 4, 263-322.

[Palmer93] M. Palmer, R. Passonneau, C. Weir, and Finin. The KERNEL text understanding system. *Artificial Intelligence*, 63(1-2):17–68, Oct. 1993.

[Sezgin01] Sezgin, Metin; Stahovich, Thomas and Davis, Randall (2001). Sketch Based Interfaces: Early Processing for Sketch Understanding. Proceedings of PUI-2001, November 2001

[Stahovich97] Stahovich, T. F. "Interpreting the Engineer's Sketch: A Picture is Worth a Thousand Constraints," AAAI Symposium on Reasoning with diagrammatic Representations II, Cambridge, Massachusetts, November, 1997.

[Weinstein01] Weinstein, E, SpeechBuilder: Facilitating Spoken Dialogue System Development, M.Eng. thesis, MIT Department of Electrical Engineering and Computer Science, May 2001.