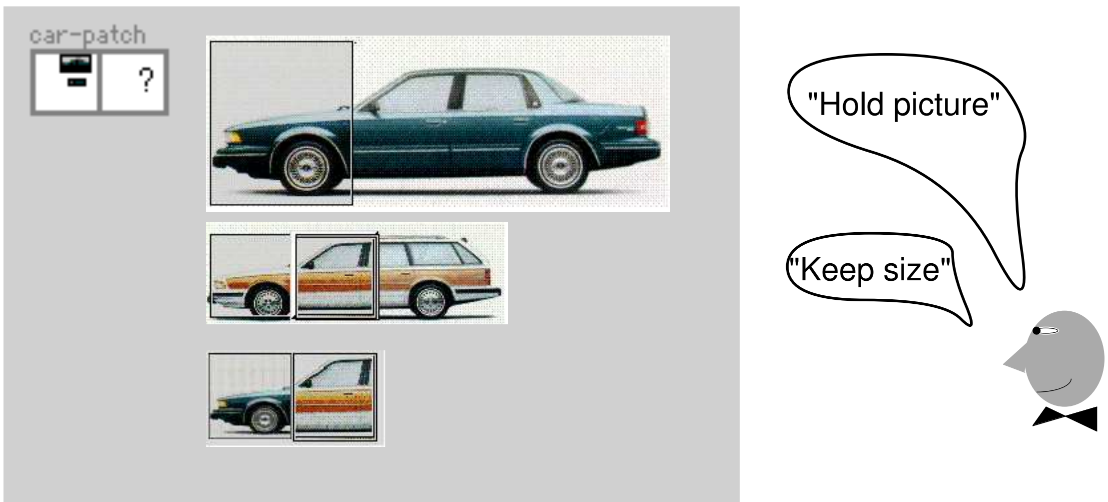
# Hearing Aid: Adding Verbal Hints to a Learning Interface


*Elizabeth Stoehr*
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
+1-617-225-6516
liz@mit.edu

*Henry Lieberman*
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
+1-617-253-0315
lieber@media.mit.edu



**ABSTRACT**

Programming by demonstration systems learn simple tasks by observing a user's manipulation of representative graphical objects. These systems then choose one meaning from many possible interpretations. Using keyboard or mouse input to select the interpretation creates an "interface gridlock" where the same input device is required repeatedly in each step. Voice input provides an independent input channel.

"Hearing Aid" is a voice input extension to the programming by demonstration system Mondrian [4]. With a mouse, the user manipulates objects in a Mac-Draw like editor while speech commands specify the appropriate generalization the learning mechanism should use. This gives the user control over the learning process while preserving Mondrian's original interactiveness and spontaneity.



**KEYWORDS**

Speech recognition, programming by demonstration, generalization, machine learning, multi-modal interfaces


**THE GENERALIZATION PROBLEM IN PROGRAMMING BY DEMONSTRATION**

Despite the great strides in user-friendliness made by modern graphical interfaces, a need remains for tools that enable non-programmers to express procedural ideas without depending on professional programmers. *Programming by demonstration* [1] is an alternative that does not require knowledge of a textual programming language to instruct a computer. The computer observes the user's actions in the interface, and generalizes those actions into a program that can later be used in similar applications.

Unlike simple macro recorders, which only play back a series of recorded keystrokes and mouse positions, programming by demonstration systems translate keyboard and mouse actions into a symbolic form that represents the user's intent. The procedure can then be applied to substantially different scenarios from the ones the computer first observed. The translation process ignores details relevant only to a specific instance, such as the exact position of objects on the screen, or size.

A basic problem for programming by demonstration systems is the inherent ambiguity of the generalization process. Many systems have heuristics that can make a good guess as to the user's intent, but this process can never be perfect. Like a human student, a learning system sometimes misunderstands the teacher.

Suppose the user defines the command *gen-ex*, using the programming by demonstration system Mondrian (Figure 1). The user modifies the original rectangle, shown at top left of Figure 1, to create an L-shape (top right). The system could use one of many interpretations when applying the *gen-ex* definition on a new rectangle (shown lower left of Figure 1). The top L is constructed proportional to the new rectangle. The middle preserves the width of the vertical leg and the height of the horizontal. In the last, (the width of first leg) : (height of second leg) ratio created in the definition is repeated in each new L. All three are valid interpretations.

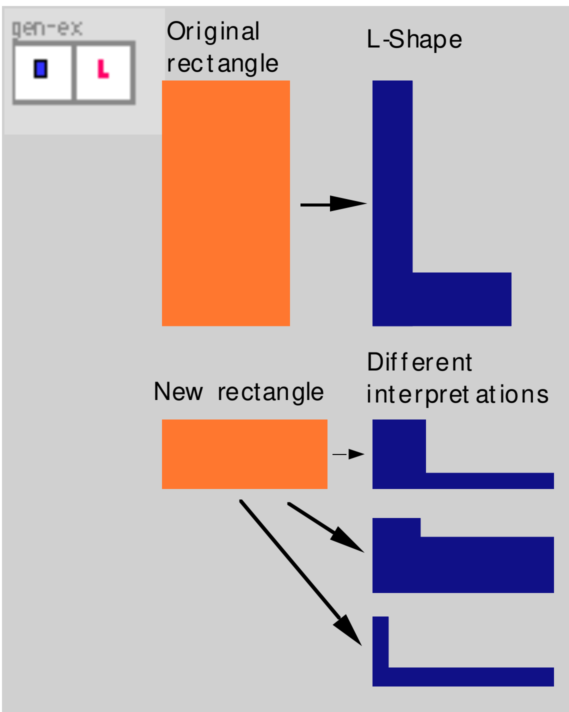



**Fig. 1**: Top: The user demonstrates how to create an L from an original rectangle. Bottom: Shows how Mondrian could potentially translate *gen-ex* into a multitude of differently shaped L's.

To alleviate this confusion, a system designer often will hardwire a single interpretation into the learning mechanism. Alternately, the user may be given the ability to choose one interpretation from a list of many generalizations. Learning systems which can produce different generalizations from the same input are said to have *dynamic bias*. This allows the same keystroke combinations to be adapted differently depending on a second command. Unfortunately, choosing an interpretation can interfere with the demonstration process itself. Past attempts to allow multiple interpretations often included large tables or menus from which a user selected a specific generalization. These menus can become so large that they are more confusing than helpful. A typical example from Kurlander's and Feiner's Chimera system [3] follows (Figure 2).

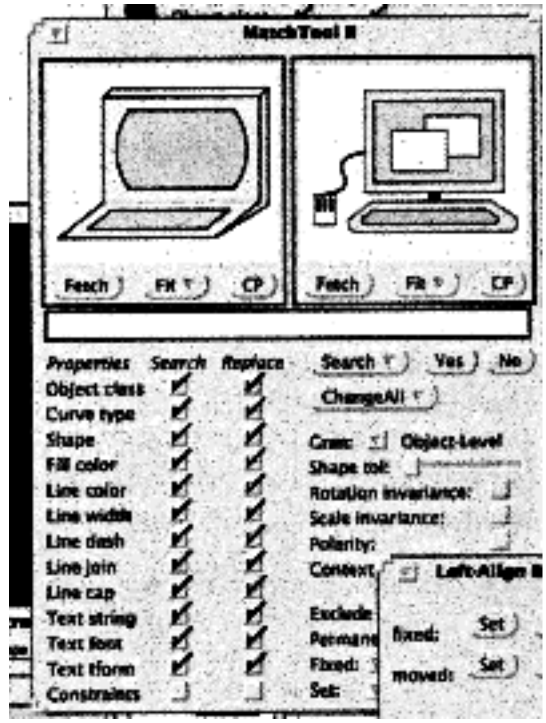



**Fig. 2**: The menu specifies different interpretations for David Kurlander's and Steven Feiner's Chimera system.

Here, the keyboard or mouse device used for design is required to select an interpretation choice from a menu. This can lead to "interface gridlock" where too many pieces of input are required through one interface. To avoid this gridlock, Hearing Aid uses speech to control its dynamic bias.

## HEARING AID

By using speech commands to specify a generalization, the user's hands are at liberty to manipulate and create objects in the graphical editor. To experiment with this idea, we have incorporated a listening mechanism into Mondrian. The specific generalizations are selected by voice through a system that has been named *Hearing Aid.* Hearing Aid first adds new possible interpretations to Mondrian's learning mechanism, and second, a means to select individual generalizations through spoken commands.

The "raw material" of the demonstration process is the low-level input of mouse clicks, text input, and mouse position descriptions. Mondrian's learning mechanism creates a symbolic description of the user's intent from this sparse listing. This process, which involves describing each object and action involved in a definition, is called *data description* by Daniel Halbert [2].

In the previous *gen-ex* example, the user begins by drawing the first leg of the L over the defining rectangle.

The pixel positions of the start and end point of the drag operation, the color, and the name are sent to a group of "Remember Functions" that generalize the data into CLOS code.

To determine size, Mondrian's generalization process depends on its interactive graphical context. When constructing *gen-ex*'s first leg, Mondrian notices that the start point of the drag is approximately the same as the upper left corner of the defining rectangle. To apply the completed definition to a new rectangle, however, the upper left corner of the new rectangle is used instead. Without Hearing Aid, Mondrian's generalization process would be "hardwired" to interpret the object's height and width are remembered as proportions of the defining object. When recalled, the definition produces an object with the same proportions to the calling rectangle's used in the defining process.

**Hearing Aid Adds Multiple Interpretations**
Hearing Aid influences Mondrian's learning mechanism by changing how Mondrian remembers objects or actions (Figure 3).

For example, the Mondrian function, "remember-point-on-object" translates a point's pixel position into a key vertex: left-top or right-bottom. Each vertex is drawn in reference to an object within the graphical editor. To determine height or width, "remember-point-on-object" first specifies a vertex on the reference object. The size is a product of the reference object's height or width and the distance between the new vertex and the reference vertex.

In one scenario, Hearing Aid adds the interpretations "maintain height", "maintain width", and "double height" to Mondrian's generalization vocabulary. These commands alter how remember-point-on-object functions. The two maintain commands remember height or width as equal to the defining object's size rather than a proportion. Independent of the size of the recalling object, "maintain as" definitions will generate objects with the same height or width used in the defining process. The double-height generalization multiplies whatever proportional distance remember-point-on-object generates by two.

**Spoken Commands Select Interpretations**
Simple vocal commands were chosen that could be applied to multiple situations (Figure 4). The commands constrain which facets of an object or action the user expects Mondrian to remember when re-applying a user's demonstration. The verbal advice is very incomplete. For example, the commands, "maintain height," "maintain width," and "double height," constrain Hearing Aid's size definition. The specific height or width attributes, however, must be specified by mouse or keyboard.

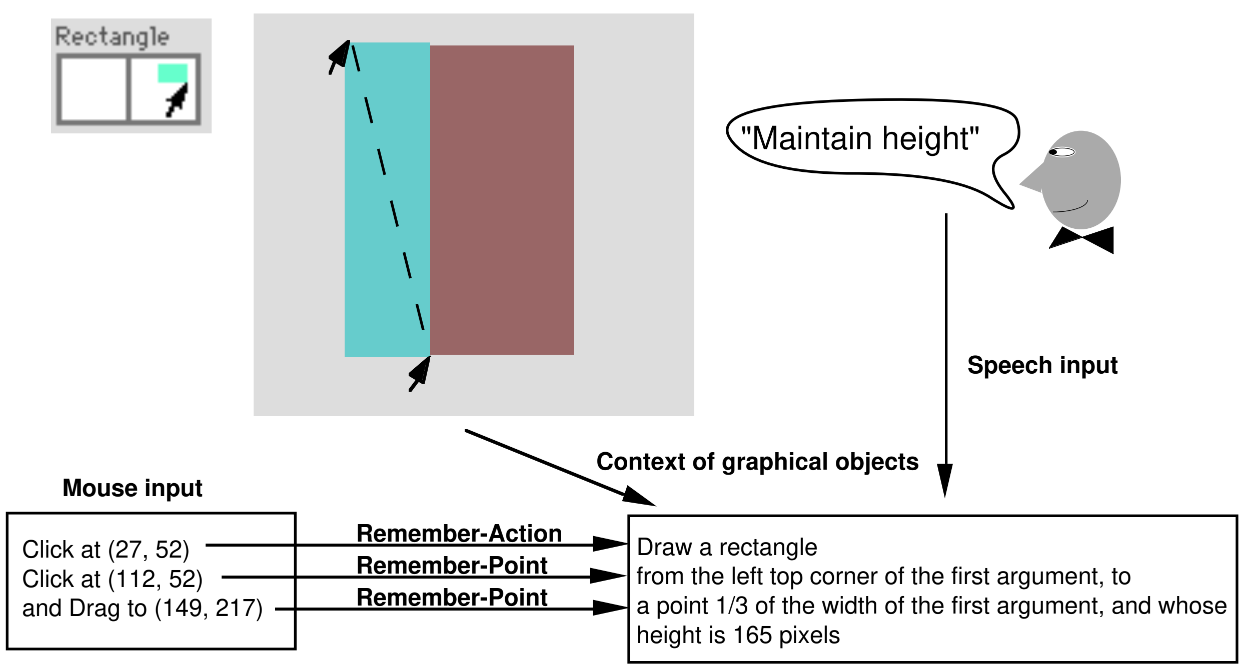
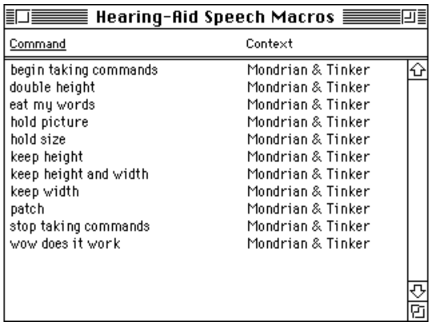



**Fig. 3**: Information from the mouse, the graphical editor, and the speech recognizer together describe the user's procedure. The Mondrian learning mechanism selects the appropriate interpretation through Hearing Aid's verbal constraints.



Hearing-Aid Speech Macros

| Command | Context |
|---|---|
| begin taking commands | Mondrian & Tinker |
| double height | Mondrian & Tinker |
| eat my words | Mondrian & Tinker |
| hold picture | Mondrian & Tinker |
| hold size | Mondrian & Tinker |
| keep height | Mondrian & Tinker |
| keep height and width | Mondrian & Tinker |
| keep width | Mondrian & Tinker |
| patch | Mondrian & Tinker |
| stop taking commands | Mondrian & Tinker |
| wow does it work | Mondrian & Tinker |

**Fig. 4**: This is one Hearing Aid vocabulary, expressed as Plaintalk speech macros. This simple vocabulary deals with size constraints.

Each verbal command is a small building block for a more precise language. Potential verbal commands such as "maintain color," "double font size", or "maintain text phrase" would also be viable and require only simple modifications for Mondrian to understand. A limitless number of verbal vocabularies is possible.

To recognize the user's vocal commands, we chose Apple's Plaintalk [Apple 92]. Plaintalk is a speaker-independent speech recognizer that stores a few pre-chosen commands. It responds to each command with a "speech macro" written in AppleScript. For Hearing Aid to function, Plaintalk needs to pass a speech macro that contains command strings to Mondrian. All that is required to create a new vocal command is to write a speech macro response to Mondrian. The new macro would contain a command that guides the Mondrian learning mechanism to follow an alternative generalization.

**Visual Markers Signpost Generalizations**

For non-proportional generalizations, markers remind the user that an alternate method is in use. The Maintain constraint displays an "M" on the rectangle while the double constraint expresses a double line symbol. This visual feedback allows users to easily revise a definition before its completion. Markers serve this purpose while avoiding the screen crowding common with displays of a procedure's history. When recalled, the markers used in the demonstration process disappear from the generated objects.

## TEST SCENARIO: INTERPRETATION OF RECTANGLE CONSTRUCTION

In the following scenario, three rectangles are constructed with identical mouse commands. However, subsequent calls produce three differently sized rectangles due to verbal input (Figure 5b).

We begin the definition by drawing a rectangle on the left side of the defining rectangle. The command is given with no speech input. Therefore, Hearing Aid directs the learning mechanism to use the default interpretation, proportional. Note that Mondrian remembers the right-bottom coordinate as (x-right-bottom, y-right-bottom) of the rectangle.

The second rectangle is drawn down the center of the defining rectangle. The user says "maintain height" which directs the learning mechanism to use the Maintain constraint. Hearing Aid alters the right-bottom coordinate to be (x-right-bottom, y-left-top + height) of the rectangle. Only this coordinate has been changed which avoids dramatically altering the learning mechanism. An "M" is displayed to express the activity of the "Maintain" constraint.

The final rectangle is constructed after saying "double height." The learning mechanism remembers the right-bottom coordinates as (x-right-bottom, (y-right-bottom * 2)) of the rectangle. Hearing Aid also displays two lines on the rectangle as a marker.

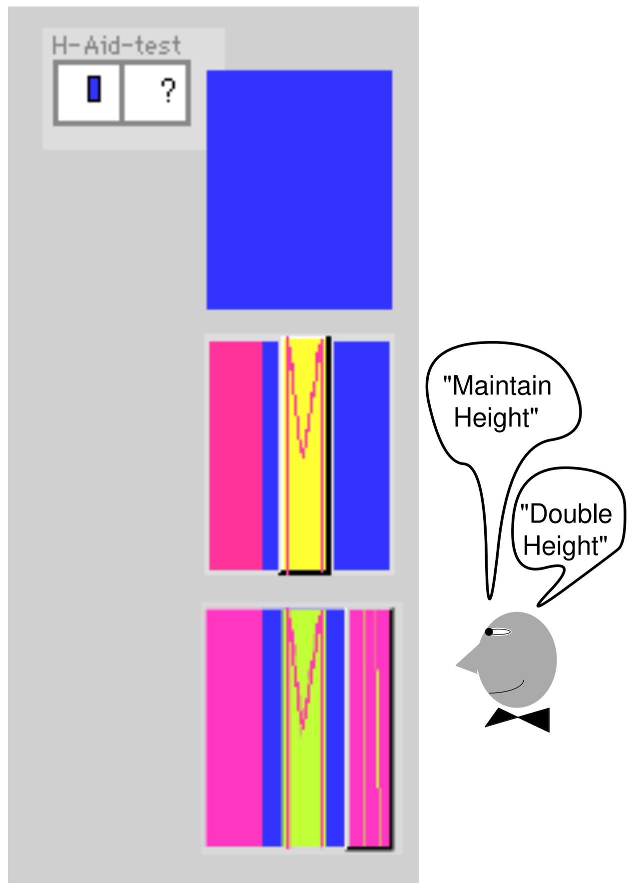



**Fig**. 5A: Top: The defining rectangle. Middle: the maintain height interpretation showing the M marker. Bottom: the double height interpretation showing the double lines symbol

The completed definition is applied to a new calling rectangle. Three rectangles of different sizes result (Figure 5B).

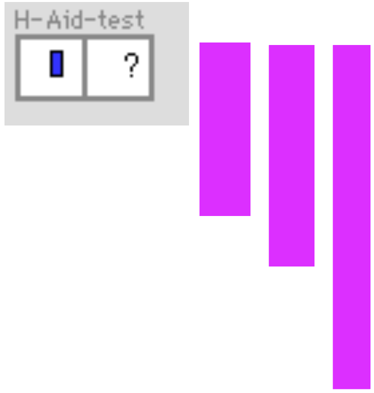



**Fig. 5B**: The completed definition is applied to a new object. The 3 generated rectangles were described by the same mouse commands but different verbal commands.

## A MORE PRACTICAL SCENARIO: PICTURE CONSTRUCTION

This scenario deals with a more realistic problem than the rectangle example. A user is given a group of differently sized car pictures. The goal is to blend separate sections of these pictures together into one uniformly sized car.

The application takes advantage of Hearing Aid's ability to store information and use multiple input techniques. If rectangle dimensions can be stored, so can size, graphics, and text information. A mouse drag operation encircles the area of a graphic from which Mondrian must glean information. A concurrent spoken command tells Mondrian which information is important. Later, the pieces of information are manipulated with a verbal command to create an entirely new picture.

Mondrian has a primitive for picture composition, called "New Part." New Part selects an area of a picture and applies rectangle characteristics to it. This "picture-part" can be moved, selected, resized etc. Hearing Aid endows New Part with the ability to store information about the picture's size or its graphic. The verbal commands, "keep size" and "hold picture" differentiate the two types of storage.

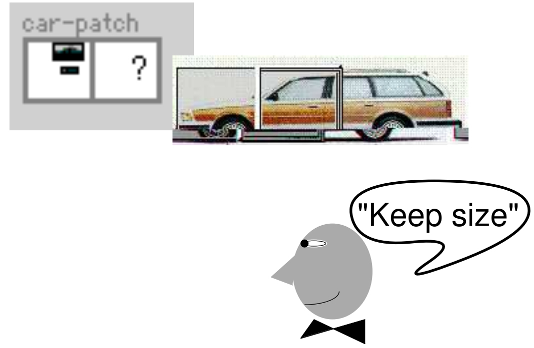



**Fig. 6**: The Hearing Aid constraint, "keep size," causes this New Part call to store the picture-part's size.

In this scenario, we will generate a "patchwork picture," that creates a new picture by merging the size and graphic aspects of the two different pictures. First, two car pictures with different sizes are selected for the definition *car-patch*. After saying "keep size," the user isolates the front of the smaller picture, the station wagon, with New Part (Figure 6). The "keep size" interpretation activates a command which records the dimensions of the selected picture part. The learning mechanism also remembers to apply the "keep size" constraint on this New Part call. When *car-patch* is recalled, this step will always store the size of the appropriate picture-part.

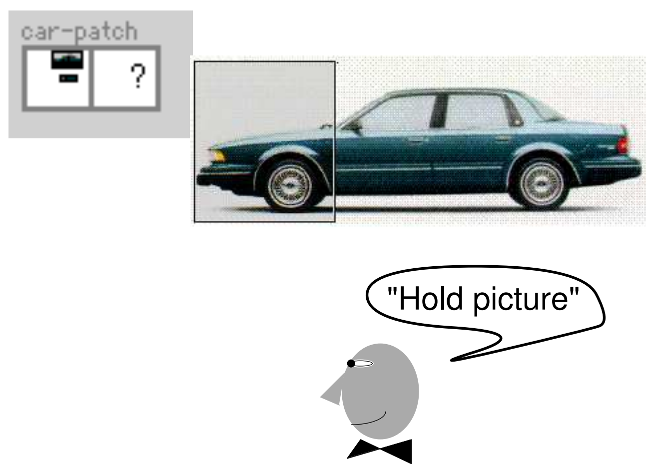



**Fig. 7**: The Hearing Aid verbal constraint, "hold picture," causes New Part to save the graphic information of this picture-part.

The user then says "hold picture" and selects the sedan car's front (Figure 7). A similar update command stores the graphic information of the picture-part. This generalization causes the learning mechanism to remember that the "hold picture" constraint must be applied to this New Part call.

To shrink the large sedan car's graphic to fit the smaller station wagon's size constraint, we added a new command to Mondrian called Patch. The Patch command is unique. It cannot be invoked without a speech command.

Each of three input methods, keyboard/mouse alone, keyboard/mouse and speech, and speech alone, is optimal in different situations. While Hearing Aid focuses on keyboard/mouse and speech dual input, Patch experiments solely with speech input. Patch tells Mondrian how to merge two pieces of information that Mondrian already has stored. A keyboard/mouse command, which normally specifies location or lists textual information, is unnecessary. Here, a speech command is not a replacement for keyboard/mouse input, but the most efficient way to signal a predefined process.

To construct the composite picture-part, the user says "patch" and clicks on some area of the defining object (title photo). The click is necessary to jump start the Mondrian learning mechanism. The patching process may be repeated to create a collage of picture parts. After saving, the definition can be called later to patch together other car pictures.

In the L-shape example, the user selects different methods of remembering length with verbal commands. Similarly, Hearing Aid alters New Part's function to include graphic information storage as well as graphic selection. Hearing Aid modifies how an action and an object is remembered by Mondrian's learning mechanism.

## PROBLEMS WITH SPEECH RECOGNITION

First, Hearing Aid is hampered by the flaws of current speech recognizers. Presently, a user must wait for Hearing Aid to recognize each spoken command, send the correct command to Mondrian, and wait for Mondrian's response. As an experimental model, the wait is not excessive, but it would be cumbersome in the workplace. New technology may be the only solution. With more sophisticated recognition programs that have larger vocabularies and quicker response times, oral commands could become more accurate and the wait less obvious.

Second, an oral vocabulary can grow too large to use easily. To foster interactive programming sessions, oral commands should be simple, so the user can recall them from his or her own memory. Narrow definitions for specific applications should be built from broad "building block" commands the user creates.

Speech recognizers have an unfortunate tendency to recognize extraneous speech. Commands need to be chosen that will not trigger a computer response by day to day "office talk." On/off switches that separate Hearing Aid from normal work times may prevent activation of unwanted interpretation commands during a design process.

## RELATED WORK

Several programming by demonstration systems have tackled the multiple generalization problem. David Kurlander's and Steven Feiner's Chimera [3] is an example based graphical editor which allows multiple generalizations for its learning process. Chimera chooses what it believes is the most likely generalization. The user may override the computer's decision by selecting the correct interpretation from a text list. A visual display of the demonstration session's history allows a user to redefine a generalization at any point in the process. Although effective, Chimera causes a user to search a list of generalization commands with a mouse, potentially causing the "interface gridlock" we were trying to avoid.

Daniel Halbert's SmallStar [2] applies programming by demonstration techniques to the office system, Xerox star. When learning a definition, it creates a data description sheet listing the patterns the learning mechanism finds important. If the user feels that it is studying the wrong information, the user can change its learning technique by updating the data sheet. Because Mondrian deals with simpler processes than office mail flow, its generalization process could be influenced in a more step-by-step fashion.

David Maulsby [5] implemented a facility for speech hints to generalization in a graphical editor, Moctec. Maulsby's system provided a rich vocabulary that included hints like "pay attention to..." and "ignore..." to indicate salience of graphical properties. This was the only other system where speech input affected the generalization process. However, Moctec was an interactive mockup rather than a full programming by demonstration system.

Closest to our work was that of Alan Turransky [6], who used speech to specify an interpretation of position, also as an extension to Mondrian. It specified where to place lines around a textual string (Figure 8, compare to Figure 3). The system was implemented with Voice Navigator, a single-word, speaker-dependent speech recognizer.

In Turransky's system, voice input immediately affected the graphical objects. The coordinates were modified before passing them to Mondrian's learning mechanism, but Turransky's system did not change the generalization process itself. While useful, it changed the raw material the learning mechanism "observed". Our goal was to guide Mondrian's learning mechanism with multiple generalizations.

## FUTURE WORK

Just as programming by demonstration systems manipulate simple graphical objects to create more complex definitions, they could potentially build more complex verbal commands. For example, one interpretation may remember a rectangle's height versus its color, or a picture's size rather than its graphic. Armed with a small vocabulary of possible constraints, Mondrian could learn more complex vocal commands such as "double maintained height" by observing the user's vocal actions. Because the interpretations would be designed by the user, the complexity could not go beyond what he or she understands. This preserves Mondrian's simplicity while giving a new adaptive ability to its learning process.

## SPEECH RECOGNITION ENHANCES (NOT REPLACES) MOUSE COMMANDS

Multi-modal systems like Hearing Aid can help a user feel more at ease with computer communication. In our interaction with other people, we are used to using both verbal and visual means of communication. Furthermore, the verbal and visual modes interact, since people talk about things they see, and show others objects they talk about. By contrast, most computer activities, such as electronic mail or programming, are limited to the single mode of typed text.

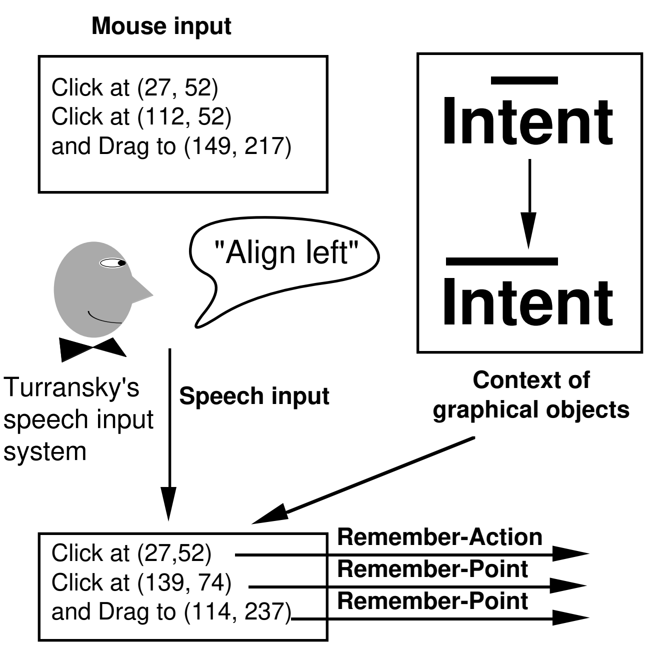



**Fig. 8**: In Turransky's vocal application to Mondrian, voice input changes mouse input rather than affecting the actual generalization process.

Even in many other applications of speech recognition that we have seen, speech and visual modes rarely interact. Speech recognition has served primarily as a substitute for typing or icon selection. With a responsive speech recognizer, this can help a user significantly. However, we are interested in the ability of voice to perform tasks that are difficult for a keyboard alone.

In addition to speech input, Mondrian also uses speech output. Mondrian contains a set of narration functions that verbally describe the user's actions during the entire definition. Combining the visual design process constrained by verbal commands with a narration of the process accented by visual markers, creates an essential verbal and visual feedback loop.

Also, supplementing a simple command with a second interface allows more complex commands to evolve from a few. The system is kept simple because both vocabularies are small. However, the language can adapt to many applications because commands are created by merging new combinations of verbal and keyboard commands. For programming by demonstration to be truly useful in the real world it needs the ability to adapt to more complex tasks, and speech recognition has proven effective in increasing the power and flexibility of the interface.


## ACKNOWLEDGMENTS

Support for this work comes in part from research grants from Alenia Corp., Apple Computer, ARPA/JNIDS, the National Science Foundation, and other sponsors of the MIT Media Lab.


## REFERENCES


1. Cypher, Allen. Bringing Programming to End Users. *Watch What I Do: Programming by Demonstration.* Allen Cypher ed. MIT Press, Cambridge, MA 1993.

2. Halbert, Daniel. Smallstar: Programming by Demonstration in the Desktop Metaphor. *Watch What I Do: Programming by Demonstration.* Allen Cypher ed. MIT Press, Cambridge, MA 1993.

3. Kurlander, David and Feiner, Steven. A History-Based Macro by Example System. *Watch What I Do: Programming by Demonstration.* Allen Cypher ed. MIT Press, Cambridge, MA 1993.

4. Lieberman, Henry. Mondrian: A Teachable Graphical Editor. *Watch What I Do: Programming by Demonstration.* Allen Cypher ed. MIT Press, Cambridge, MA 1993.

5. Maulsby, David. Instructible Agents. PhD Thesis. University of Calgary, 1994.

6. Turransky, Alan. Using Voice Input to Disambiguate Intent. *Watch What I Do: Programming by Demonstration.* Allen Cypher ed. MIT Press, Cambridge, MA 1993.