Natural Sketch Recognition in UML Class Diagrams

Tracy Hammond HAMMOND@AI.MIT.EDU
MIT Artificial Intelligence Laboratory, 200 Technology Square, Cambridge MA, 02139 USA

1. Introduction

Sketching is a natural and integral part of software design. Software developers use sketching to aid in the brainstorming of ideas, visualizing programming organization, and understanding of requirements. Unfortunately, when it comes to coding the system, the drawings are left behind. Natural sketch recognition offers a way to bridge this gap.

I created a natural sketch recognition environment for UML (Unified Modeling Language) (Alhir, 1998). My system differs from graffiti-based approaches to this task, in that it recognizes objects by how they look, not by how they are drawn. My goal is a system where the user can sketch UML diagrams on a tablet or whiteboard in the same way they would on paper, but the diagrams would then be recognized by the computer to provide clean interpreted diagrams, stub code, and enhanced editing ability.

I selected UML diagrams because they are a de facto standard for depicting software applications. Within UML I focused on class diagrams, first because of their central role in describing program structure, and second because many of the symbols used in class diagrams are quite similar. Hence they offer an interesting challenge for sketch recognition.

2. A Class Diagram Recognition System

2.1 Class Diagrams within UML

Class diagrams describe the static structure of a system, how it is structured rather than how it behaves (Alhir, 1998).

Class diagrams consist of (i) general classes, (ii) interface classes, and (iii) associations that can exist between two classes. In UML, a general class is represented by a rectangle, while an interface class is represented by a circle or rounded rectangle.

There are three types of associations: (i) A dependency association exists if one class calls a method from another class, including the constructor. The dependency relationship is represented by an arrow with an open head. In Figure 1, the Game class is dependent on the Graphics class. (ii) A generalization or inheritance association exists if one class is a kind of or extension of another class. The inheritance relationship is represented by an arrow with a triangular head. In Figure 1, the Hand class is inherited from the CardDeck class. (iii) An aggregation association exists if one class is part of another. The aggregation relationship is represented by an arrow with a diamond head. In Figure 1, the Card class is part of the CardDeck class.

2.2 Recognition of Classes and Associations

Gestures – collections of strokes – are recognized as either creation, deletion, or movement of a class diagram glyph. The system can recognize glyphs indicating general classes, interface classes, dependency associations, inheritance associations, and aggregation associations. A stroke may also be left as unclassified. In Figure 1, the left screen displays the strokes drawn to create the interpreted UML diagram in the right screen.

Recognition is based on a combination of temporal, locality, and contextual information. Gestures to be recognized are restricted to those temporally close as people tend to sketch complete objects before sketching the next. Strokes are reduced to line segments. Angles and distances between segments are examined to help determine the most appropriate interpretation. Contextual information is used in a variety of ways. To give one example, a line drawn from one class to another may be recognized as an association but a free standing line may be left unclassified.

A general class can be drawn with one to four strokes. A collection of strokes is classifiable as a general class if the majority of the points fall between the bounding box of the strokes and a slightly smaller internal bounding box. A stroke is classifiable as an interface class if the least squares error is small (Sezgin, 2001). [1]

Arrows can be drawn as complete arrows or as a line connecting one class to another. To recognize an arrow, the recognizer first attempts to locate the head and tail of the arrow by finding the points furthest from each other (i.e., it

[1] The least squares error is the sum of the squares of the distance from each drawn point to the ellipse defined by the bounding box of the sketched points.

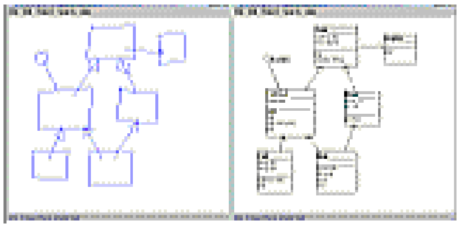

Figure 1. The left screen displays the hand drawn UML class diagram. The right screen displays the recognized output of the hand drawn UML class diagram. Enlarged figures are shown in the Appendix.

assumes the head or the tail to be a stroke endpoint). The arrow is divided into line segments. The head is defined to be the stoke endpoint closest to the other line segments. The algorithm then locates the arrowhead's side points by finding the points furthest from the head-tail line, and lying on either side of this line. The line segments are then examined to determine where they fall to determine the association type.

2.3 Recognition of Editing Commands

The system is non-modal: users can edit or draw without having to give any explicit advance notification. One editing action is moving classes and associations on the screen. The system understands a gesture as a move command rather than a drawing command based on the user's sketching behavior: Users tend to click and hover over a class when moving it. For example, the system interprets a hover of larger than .5 second as a move command. The move command is signified to the user by a cursor changing to a gripping hand with which the user can move the class. The user can delete a class or association by scribbling it out.

3. Previous Work

Work at Berkeley by Hse et al. (1999) has shown that users prefer a single-stroke sketch-based user interface to a mouse-and-palette based tool for UML design. Hse performed a Wizard of Oz experiment comparing the two design methods. During the experiment, users requested more sketching flexibility, such as the ability to draw with multiple strokes.

One company – Ideogramic (Damm et al., 2000) – has developed a gesture based diagramming tool, Ideogramic UML^{TM}, which allows users to sketch UML diagrams. The tool is based on a graffiti-like implementation and requires users to draw each gesture in one stroke, and in the direction and style as specified by the user manual. One consequence of the stroke limit is that some of the gestures drawn only loosely resemble the output glyph. For example, φ is the stroke used to indicate an actor, drawn by the system as a stick figure.

Edward Lank and others at Queen's University have developed a system to recognize sketches of UML diagrams using a distance metric (Lank et al., 2000). Each glyph (square, circle, or line) is classified not based on what the glyph looks like, but rather the total stroke length compared to the perimeter of its bounding box (e.g., if the stroke length is approximately equal to the perimeter of the bounding box, it is classified as a square).

4. Conclusion

I have described a system to recognize UML class diagrams sketches. Users are allowed to draw as they would naturally, without any graffiti-like restrictions. Sketches are recognized based on what the drawn object looks like rather than how it is drawn.

Future system enhancements include allowing the user to sketch more detail about a program. For instance, I plan to add the ability to recognize multiplicity relationships by noting properties sketched around associations.

My future plans are to connect the system to a CASE tool such as Rational Rose to enable the programmer to use the functionality provided by a CASE tool.

Acknowledgements

The author would like to thank Dr. Randall Davis for his feedback and support throughout the research. The author would also like to thank Louis-Philippe Morency, Dr. Raghavan Parthasarthy, Dr. Jan Hammond, and John Zeigler for their helpful comments and suggestions.

References

Alhir, S. S. (1998). *UML in a nutshell: a desktop quick reference*. Cambrigde, MA: O'Reilly & Associates, Inc.

Damm, C. H., Hansen, K. M., & Thomsen, M. (2000). Tool support for cooperative object-oriented design: Gesture based modeling on an electronic whiteboard. *CHI 2000*.

Hse, H., Shilman, M., Newton, A. R., & Landay, J. (1999). Sketch-based user interfaces for collaborative object-oriented modeling. Berkley CS260 Class Project.

Lank, E., Thorley, J. S., & Chen, S. J.-S. (2000). An interactive system for recognizing hand drawn UML diagrams. *Proceedings for CASCON 2000*.

Sezgin, M. (2001). Early processing in sketch understanding. Submitted to *Proceedings of the International Joint Conference on AI*.

Natural Sketch Recognition in UML Class Diagrams

Tracy Hammond HAMMOND@AI.MIT.EDU
MIT Artificial Intelligence Laboratory, 200 Technology Square, Cambridge MA, 02139 USA

Appendix

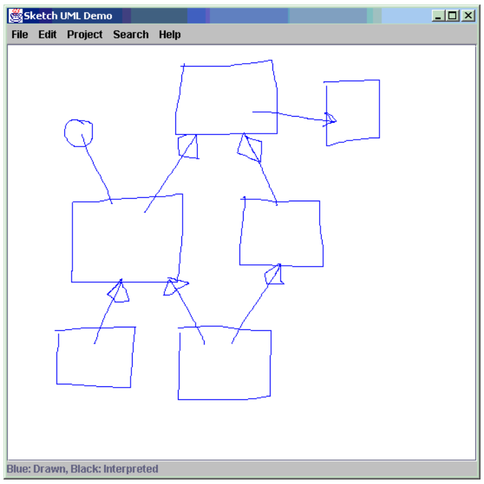

Figure 2. Hand drawn UML class diagram.

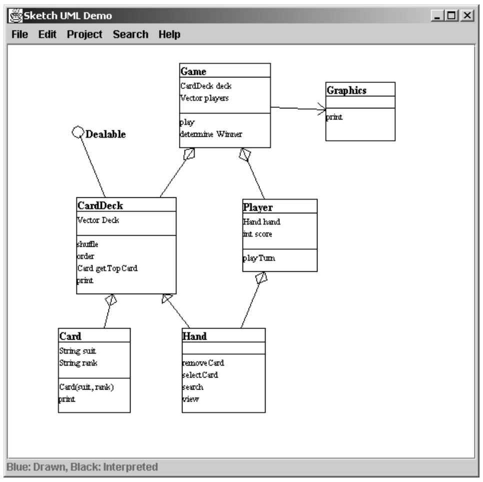

Figure 3. Recognized output of UML class diagram in Figure 2. Note that the letters shown are typed in through a pop-up screen and not recognized.