

SketchUML – Sketch based approach to Class Diagrams

Gaurangi Tilak

Computer Science Department,
University of California,
Santa Barbara
gaurangi_tilak@cs.ucsb.edu

Krithika Ananthkrishnan

Computer Science Department,
University of California,
Santa Barbara
krithika@cs.ucsb.edu

ABSTRACT

We have designed and implemented SketchUML, a sketch based recognition system for class diagrams in UML. UML diagrams form an important part of the software design process. Using SketchUML we would like to create a tool that provides a more natural interface to designers for generating class diagrams. SketchUML is based on recognizing fragmented strokes using their geometrical properties. This lets the user draw freely without any conditions on the user's style of drawing. The system has two viewing options, drawn view and cleaned up view allowing the users to take advantage of the entire functionality while viewing their original drawn diagrams or the cleaned up versions. The system also provides the option of generating a skeleton code based on the drawn diagram along with some basic UML structural validations. The evaluations showed that users were comfortable using SketchUML to create class diagrams.

Author keywords

Sketch recognition, UML diagram, User interfaces

ACM Classification Keywords

H.5.2 [User Interfaces] Input devices and strategies, D.2.2 [Design Tools and Techniques] Computer-aided software engineering (CASE)

INTRODUCTION

The design of any system is the result of a creative process. A designer's most preferred aid is a pen and paper with which he can freely work on his design without any restrictions. Thus, sketching forms an integral part of the design process. In software design too, the first draft of the system design is sketched on a paper. This design then has to be converted to a more formal format i.e. it has to be put into a design tool.

Traditional computer aided software design and engineering (CASE) tools do not allow free from input while creating the UML diagrams. In SketchUML we have created a tool that can combine the flexibility and ease of paper style

sketching with the power of a design tool with automatic code generation. From all the different diagrams within UML we have chosen class diagrams as the representative diagram for recognition. From the perspective of sketch recognition, they provide considerable challenges for recognizing and interpreting the design diagram.

RELATED WORK

Due to the importance of UML diagrams in the field of software design, a number of systems have been developed to recognize and interpret sketched UML diagrams.

Lin Qiu built SketchUML [1], a UML diagram recognition system that allows students to create UML class diagrams on a flexible paper-like interface to facilitate learning-by-doing design exercises. It provides various drawing, editing and text recognition functionalities. We have incorporated the geometry-based approach of sketch recognition. Using this approach we recognize sketches by the shape and relative location between strokes rather than how the strokes are drawn. This allows users to draw shapes naturally with any number of strokes and in any order.

Tahuti developed by Tracy Hammond and Randall Davis [4], is a geometrical UML sketch recognition system. It is based on a multi layer recognition framework, which recognizes multi stroke objects, by their geometrical properties allowing users the freedom to draw naturally. The system also provides the users with dual viewing possibilities where the sketch can be viewed as it was drawn by the user or as a cleaned up version. The low level recognition system that we have implemented is the same as the one used in Tahuti, which is based on feature point detection using curvature and speed data. (Sezgin, Stahovich and Davis 2001) [2].

SOFTWARE ARCHITECTURE

Technology

SketchUML has been implemented using C# and .NET Framework 3.5 as a Windows stand alone application. We have used the Microsoft Tablet PC SDK for tablet input support. It provides basic ink collection and handwriting recognition capabilities.

System Overview

SketchUML system comprises of three main modules

- User Interface module – Menus, Drawing area

- Sketch recognition module – Classes for all the sketched shapes, lines, strokes, etc.
- Code generation module – Classes for implementing the diagram validation and code generation functionalities

The sketch recognition module is the most important module in the system. It includes all the classes that represent the various sketched shapes, lines, strokes, etc. These classes form the core of the recognition process. The class hierarchy has been implemented in such a way that all the recognized shapes are sub classes of a common generalized shape class. This class hierarchy helped in generalizing the functionalities that were common for all shapes. With this class hierarchy it is very easy to find out the topology of the sketched class diagram, which is useful for a number of functionalities such as code generation, diagram validation, save, etc. Also, this hierarchy makes the system scalable i.e. it would be easy to add more shapes into the hierarchy without affecting the rest of the system.

SYSTEM IMPLEMENTATION

Recognition process

The core part of any sketch recognition system is recognizing and identifying the strokes. The simplest system will need a user input to indicate that the sketch is complete and start recognizing it. We have implemented a modeless recognition system that recognizes the shapes as the strokes are drawn. This provides users the freedom to draw without having to think about the recognition process. We also try to put as few restrictions on the user's drawing style as possible. Feedback regarding the recognized shape is provided as soon as the shape is recognized, but only as a coloring of the shape. We feel that keeping the feedback minimum and non-intrusive will make sure that it does not interfere with the design process.

Currently, our system can recognize the following symbols that are used in UML class diagrams –

- Class symbol (represented as a rectangle)
- Interface symbol (represented as an ellipse)
- Connections between the class and interface
 - Simple association connection (a line connecting the two symbols)
 - Generalization – Specialization connection (a line with a closed triangular arrowhead connecting the two symbols)

The recognition process for the sketches is implemented at two levels – low level recognition performs basic processing on the stroke and fragments it into a series of lines, the high level recognition process combines these lines to recognize UML specific shapes. The domain of class diagrams does not use any shapes with curves except an ellipse, so even before low level recognition, we perform

ellipse recognition. If a stroke does not get classified as an ellipse then we can assume that this stroke consists of a series of lines and proceed with the recognition.

Ellipse Recognition

The first step in the recognition process is to classify whether a stroke is an ellipse or not. It has been observed that generally people tend to draw an ellipse using a single stroke. We have used this observation and restricted the user to drawing ellipses using only a single stroke. To perform ellipse recognition, we calculate the approximate distance of each stroke point from the best fit ellipse in the stroke's bounding box. The squared error for the stroke is calculated and a stroke is classified as an ellipse if this error is less than a specific threshold. As we are performing ellipse recognition before recognizing any other shapes, it is important that we have as less false positives as possible. We have chosen an error threshold that is dependent on the length of the stroke and experimentally refined it to give the optimum ellipse recognition. However, it is a hard problem to choose a threshold that will give a perfect classification.

Low Level Recognition

For the purpose of recognizing a stroke as a series of lines, all the corners in the stroke are detected first. We have implemented a corner detection algorithm based on the algorithm developed by Sezgin, Stahovich and Davis [2]. This algorithm uses the curvature and speed data of the stroke to come up with possible corner points. To classify points as corners the local extrema are found for portions of the curvature and speed data that lie beyond a certain threshold. False positives due to noise are present in the corners detected using curvature data as well as speed data. To avoid selecting these false positives we use the corners present in a hybrid fit generated from the curvature and speed corners based on a certainty metric for the corners.

After we detect the corners in the stroke, we create lines by connecting the consecutive corners. To refine the corner selection and get rid of false corners, we merge the lines based on their slopes i.e. the consecutive lines having similar slopes are merged into a single line. We also merge lines with very small lengths as they generally represent erroneous corners. As the processing of each stroke is done immediately after it is drawn, we also check if newly created lines can be merged with any of the previously drawn lines. Once the stroke has been fragmented into lines, all the lines drawn till now are scanned and lines are

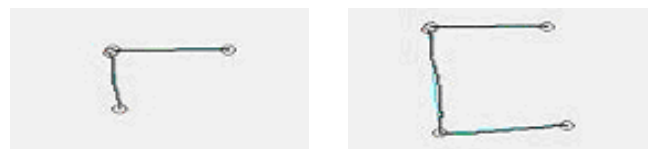


Figure 1: Left - First stroke drawn, Fragmented into two lines; Right – Second stroke drawn, Vertical lines of the two strokes are merged as they are connected and have similar slopes

merged if they are spatially close and have similar slopes. (See Figure 1)

High Level Recognition

The high level recognition process uses the lines fragmented by low level recognition and combines these lines to recognize specific shapes. For recognition, we use an algorithm that performs a recursive greedy search through all the fragmented lines to combine the lines into forming the shapes. Starting with a line, the algorithm traverses the unrecognized lines to form a tree of connected lines. It continues building the tree until it finds the required shape or until there are no further connecting lines. The condition that terminates the algorithm depends on the shape that is being recognized. The termination conditions for the different shapes are –

- Rectangle - There have to be four connected lines forming a closed shape and satisfying the basic geometrical requirements of a rectangle i.e. adjacent lines have to be perpendicular and the opposite lines have to be parallel
- Simple association connection - There have to be two distinct symbols (rectangle or ellipse) at the two ends of the connection
- Triangular arrowhead for generalization-specialization connection - There have to be three connected lines forming a closed shape

The generalization-specialization connection is a simple association connection with a closed triangular arrowhead.

Erasing

The user can erase the strokes that have been drawn. We have implemented the erase functionality to erase a line or a shape. A user using the tablet with a pen as the input device can use the back tip of the pen to erase the strokes. The strokes that intersect the back tip stroke will get erased. If the erased line is a part of a shape, then that shape will also be removed i.e. it will disintegrate into its constituent lines.

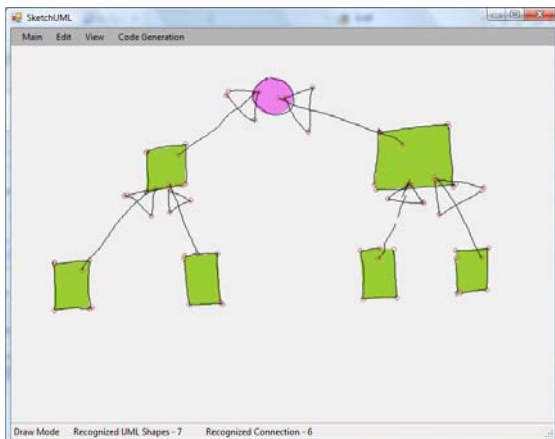


Figure 2: Drawn view of a UML diagram

Another way of erasing is by clicking the barrel button (right clicking the mouse). This opens up a context menu that has the erase option. If the barrel button is clicked on top of a shape, then the erase option will erase the entire shape. If it is clicked on top of a line, then only that line is erased and the associated shape, if any gets unrecognized. There are also two options for erase in the edit menu.

Views

We have created two possible views for the sketched diagram – Drawn view (Figure 2) and Cleaned up view (Figure 3). The user can toggle between these two views using the view menu. The Drawn view shows the sketch as the user has drawn it without any modifications. Thus, it shows the original drawn strokes and also the fragmented lines. The recognized symbols are indicated by filling the symbol with a color. The Cleaned up view replaces all the user strokes with the appropriate lines or symbols. The recognized symbols are drawn in the same locations as the user drawn symbols. Also, the line fragments that have not been recognized as yet are shown as lines in the cleaned up view. All the functionalities such as actual drawing, erasing, code generation, etc. work in both the views.

Adding Details and Code Generation

For a UML class diagram to be really useful it is necessary that the user can associate each class or interface symbol with meaningful details. In SketchUML, the user can add details to a class or interface by using the context menu option of ‘Add details’. This option opens an ink form (see Figure 3) where the user can write the various details associated with the class or interface i.e. the name, methods and properties. The boxes on this form are ink boxes i.e. the user can write in them with the pen and the handwritten text will get recognized. The user can write multiple methods or properties, each new one being on a separate line. Once the user associates these details with the symbol, the name of the symbol is displayed in the diagram. It is also possible for the user to edit these details, using the context menu.

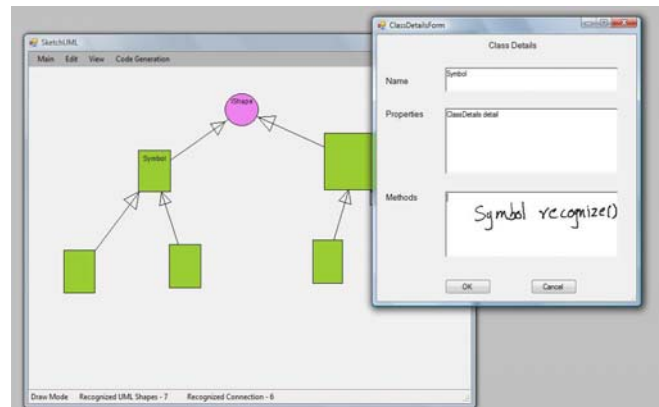


Figure 3: Cleaned up view of the UML diagram with names and the class details ink form

```

using System;
using System.Text;
namespace SketchUML
{
    class Symbol : IShape
    {
        ClassDetails detail;
        symbol recognize ()
        {
        }
        //methods inherited from interface
        void draw ()
        {
        }
    }
}

```

Figure 1: Sample of the C# skeleton code generated for the Symbol class that implements the IShape interface

After adding details to the class diagram, the user can generate a skeleton code. Before code generation, the class diagram is checked for some basic structural UML requirements. The following validations are performed –

- All classes or interfaces should have details associated to them.
- All classes or interfaces should have a name associated to them and the name should be unique.
- An interface cannot be connected by a generalization-specialization connection to a class such that the interface is extending the class.
- A class cannot inherit from two super classes.

All the issues that are found from these validations are displayed to the user with an option of continuing ahead with the code generation, ignoring the issues.

While generating the code, the skeleton of the code for each class and interface is created as one .cs file per class or interface. The code that we are generating is in C#, but the functionality can be extended to generate code in any object oriented language.

EVALUATION

We have conducted an informal evaluation of SketchUML. It was evaluated by computer science students who had some basic background knowledge of UML class diagrams. We gave a short demonstration of the system to these students instructing them about the use of the various interfaces and functionalities of the system. The students were asked to complete a survey regarding their experiences with using SketchUML. We found that the students who had not used the tablet before, initially took some time to get adjusted to the tablet. Once they had got used to the tablet, the students found the interface of the system intuitive and easy to use. We observed that users were having trouble having their drawn rectangles recognized. The students felt that the rectangle recognition was too restrictive and needed to be more flexible. We will try to improve the rectangle recognition by refining the

threshold that we use when restricting the geometric conditions for the rectangle. We also observed that some of the students preferred drawing in the cleaned up view mode while others preferred the drawn view. In both the views, the students appreciated the feedback that they got from the system about the recognition. They especially liked the user interface saying that it was neat and easy to use. Some also liked the concept of using the barrel button to erase and not having to double tap or change the pen tip. They found the code generation and the handwriting recognition part "cool". Although not a formal evaluation, this study gave us some insight into how users liked our system and how they used it. In future we would like to do a formal evaluation of the system comparing it against conventional CASE tools.

CONCLUSION AND FUTURE WORK

We have designed and implemented SketchUML, a dual-view, multi-stroke sketch recognition environment for UML class diagrams. Sketch recognition is performed using a geometry based approach to recognize class diagrams, interfaces and connections between them. An informal evaluation showed that the users enjoyed the intuitive and natural user interface provided by SketchUML.

In the near future, we would like to improve the system by making the recognition more robust and recognizing a larger range of symbols. We would like to add editing facilities such as moving the symbols or connections. We would also like to give the users an option to add details associated to connections. Another important and necessary functionality from usability point of view is the option to save the diagram and open saved diagrams. We would like to implement this functionality based on the XMI standards so that additionally we can provide the feature of exporting the diagrams to conventional CASE tools.

ACKNOWLEDGEMENTS

This project was done as a part of CS290E – Sketch based systems course. We would like to thank our professor Tim Sherwood, Ryan Dixon and all the students in the course.

REFERENCES

1. Qiu, L. SketchUML: The Design of a Sketch-based Tool for UML Class Diagrams. In *Proceedings of World Conference on Educational Multimedia, Hypermedia & Telecommunications*, June 2007
2. Sezgin T., Stahovich T, and Davis R. Sketch Based Interfaces: Early Processing for Sketch Understanding. *Workshop on Perceptive User Interfaces*, Orlando FL 2001
3. Tenbergen, B., Grieshaber, C., Lazzaro, L., Buck, R. (2008). SketchUML - A Tablet PC-based e-Learning Tool for UML syntax using a Minimalistic Interface. *Proceedings of IADIS International Conference Mobile Learning 2008* Algrave, Portugal, IADIS Press pp. 84-91
4. Tracy Hammond and Randall Davis. Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. In *AAAI Spring Symposium on Sketch Understanding*, pp.59-68. Stanford, California, March 25-27 2002