An E-whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams

Qi Chen

This thesis is submitted in fulfilment of the requirements for the degree of Master of Science in Computer Science, completed at The University of Auckland

July 2003

Abstract

The whiteboard is one of the most common tools used for supporting various activities in many fields. These include its use by software developers for doing collaborative work, especially at the early design stage. Due to the advancement of the E-whiteboard technology, much research has been carried out in this area over recent years attempting to retain the advantages of conventional whiteboards while mitigating their disadvantages. However, not all related issues have been addressed, nor sophisticated solutions have been provided in those proposed applications.

In this thesis, we present our investigation of developing an E-whiteboard based sketching tool to support the early-stage UML design. The prototype tool allows designers to sketch UML software design diagrams and free-hand annotations. A key novelty of our approach is the preservation of the sketches and the support of the pen action based manipulations to the sketches. Recognized UML sketches are formalized to computer drawn UML diagrams to provide the direct feed-back to the user, and the formalized UML diagrams can be exported to a 3rd party UML CASE tool.

The design and implementation issues are discussed in detail in the thesis. Some unique approaches and techniques we adopted in the development to achieve required tool functionality are illustrated. We also present an evaluation of our tool showing its advantages and disadvantages comparing to the conventional whiteboard and other UML CASE tools. Finally we provide our comments on the research and highlight the potential future work in this area.

Acknowledgement

I am greatly indebted to my thesis supervisor Prof. John Grundy for kindly providing guidance throughout the development of this study. His comments have been of greatest help at all times. Gratitude also goes to my second thesis supervisor Prof. John Hosking, head of the department, for providing me constant encouragement and various suggestions about the research work.

I extend my sincere gratitude and appreciation to many people who made this thesis possible. Special thanks are due to all persons participated in the survey, for the time and energy they spent which have added great values to this thesis.

I am grateful to my wife Xiaoling for the inspiration and moral support she provided throughout my research work. Without her loving support and understanding I would never have completed my present work. Particularly, I owe to my daughter Ruyan for her silent prayer for my work at the time when she needed my company most.

Finally, I would like to thank all whose direct and indirect support helped me completing my thesis in time.

Table of Contents

CHAPTER 1 - Introduction

1.1 Introduction

The whiteboard is one of the most common tools used by software developers when doing collaborative work, especially at the early design stage [Apperley et al 02, Chen 03, Guimbretiere et al 01]. For example, to sketch software requirements and design ideas, explore architectural solutions, capture high level code fragments, organize design teams, schedule events, etc [Damm 00]. Notation based modeling languages like the *UML* (Universal Modeling Language) [UML 03], are often used to assist these activities.

Figure 1.1 shows a whiteboard in use by designers (a), and an example of sketched UML diagrams on a whiteboard (b). Three UML diagram types have been sketched on the whiteboard – (1) “Use Case” diagram (stick figure and oval), describing actors (users) interacting with a system; (2) “Class” diagram (box with horizontal lines inside and arrowed/non-arrowed lines between), describing classes of objects and their relationships; and (3) “Sequence” diagram (boxes with vertical lines underneath and horizontal arrowed lines between), denoting message sequence flow between objects. There are also some secondary notations (in red and green colors) in the sketch.

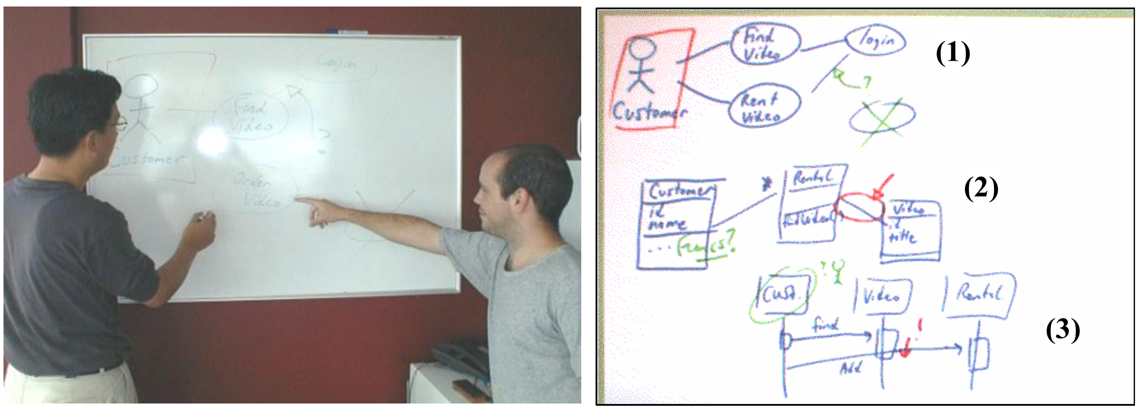


Figure 1.1 (a) Designers around a whiteboard; (b) an example of sketched UML diagrams

Research on developing software tools to support applications using large *E-whiteboards* (electronic whiteboards) has emerged over recent years [Damm et al 00, Landay 96, Perlimmer et al 02]. This is due to the recent advancement of E-whiteboard technologies [Apperley et al 02, MIMIO 03, SMART 03] and experiences gained from pilot studies on pen-based computing. The aims of these approaches are not just to replace common input/output devices (computer monitor, mouse/keyboard, etc.) with special devices (large display screen, electronic pen, etc.), They also include many other goals, e.g. providing pen-input retrieval, free hand-drawing/writing recognition, integration with conventional *CASE* (Computer Aided Software Engineering) tools, aspects in human-computer interaction, etc. Neither 'best solutions' nor 'standards' have been determined to address all of these issues as yet.

Based on the above facts, this research was undertaken to investigate an E-whiteboard application to support sketching of UML diagrams for early-stage software designs. In this thesis, we discuss our thoughts, findings and solutions documented in the development of a prototype tool for this application.

1.2 Motivation

Some key advantages of using conventional whiteboards for sketching such UML and other designs include:

- *Immediacy*: there is very little effort required to make a whiteboard "available", and it is very easy to create diagrams, capture text, and delete or extend captured information

- *Versatility*: a whiteboard can be used to sketch diagrams of multiple (even mixed) notations, as well as supporting a variety of secondary notations, such as comments, arrows, highlighting, and color. Sketches do not have to be precise nor complete in any formal manner.

- *Flexibility*. Users can use their own design style and can adapt conventions, such as the UML, to suit their own needs.

- *Size*: a whiteboard is generally big enough to hold several significant sketches and to allow several people to easily collaborate.

- *Collaboration*: a whiteboard allows multiple designers to gather around and discuss evolving designs, including taking turns at sketching and annotating designs on the whiteboard

However, there are some disadvantages of using conventional whiteboards for such design tasks:

- *A lack of persistency:* old sketches will have to be erased when there is no drawing space left on the whiteboard. There is no support for storing the history of drawings where previous sketches can be retrieved later to be reviewed or modified.

- *An inability to readily transfer information to electronic design tools:* the design work on the whiteboard has to be repeatedly entered (using keyboard, mouse or other input devices) to electronic design tools (e.g. CASE tools) for further design and implementation.

- *Very high viscosity for some changes:* e.g. repositioning, resizing or copying parts of diagrams is not possible, which increases the inefficiency when working on the whiteboard.

- *Difficulty in collaborating at a distance:* it is difficult for designers to collaboratively work on more than one whiteboards which are in separated rooms.

Our motivation was to investigate the use of large E-whiteboards to support early design stage UML sketching. We wanted to retain the advantages of conventional whiteboards while mitigating their disadvantages. The evolution of large-display devices like E-whiteboards and the availability of related mature hardware/software products in the current market have made this investigation feasible.

1.3 Goal

We have described our goal to build an E-whiteboard based early software design tool. This tool allows UML diagrams to be sketched, recognized and integrated with conventional UML CASE tools. During the development of the tool, we investigated important issues related to this area. We sought answers to questions like: what is the theoretical basis of pen-based computing? How can an E-whiteboard best be used as a special input/output device? What are the differences between an E-whiteboard and conventional input/output devices? What is the best approach for capturing and recognizing sketches or UML design elements? How can we integrate our tool with other UML CASE tools? Do we have a better recognition algorithm than others that have been proposed? What considerations need to be taken in terms of usability which suits the nature of hand-writing/drawing?

In addition, the UML sketching is an initial experimental example we choose for our investigation. It is our ambition that, upon the success of this research, we may carry out future works to extend our tool to other design areas. Thus, it is important for us to add a high level of extensibility to the tool when designing its architecture and adopting ‘best’ mechanisms and techniques for implementation.

We hope this thesis is valuable in providing thoughtful ideas, theoretical basis and practical experiences to other researchers who have similar interests in this area, as well as to ourselves for doing future works.

1.4 Research Approach

After defining the motivation and goal in the initiation of this research, we first perform a comprehensive literature review and in-depth background study prior to the development of the tool to gain solid theoretical basis for our application and to prevent us ‘reinventing wheels’. This review and study covers topics like pen-based computing and E-whiteboard technologies, conventional UML CASE tools, software design concepts, hand-writing/drawing recognition, and other related research.

The second stage is an investigation through the development of our desired tool. The tool development follows a typical, but scaled down (due to the time restriction) *SDLC* (Software/Systems Development Life Cycle). We first outline essential requirements and specifications at a conceptual level for building up such a tool. Then based on the requirements, specifications and knowledge gained from the literature review and background study, we perform the design and implementation tasks. All related research issues are investigated during this process.

Finally, we carry out a case study to illustrate the features of our tool, and an evaluation to assess the achieved functionality and the advantages/disadvantages of the implementation of our tool. Then, we provide a conclusion to summarize our research and to state the potential future work.

1.5 Thesis Overview

We have organized this thesis in seven chapters:

- *Chapter 1*: Introduction – introduces the project initiation and the thesis topic. States the motivation, goal and approach of this research.

- *Chapter 2*: Background Study – contains basic concepts and other related research from the literature study on the subject area.

- *Chapter 3*: Requirements Specification – defines the requirements specification and presents a conceptual object model of our tool.

- *Chapter 4*: Design and Implementation – discusses all design issues (e.g. architecture, data structure, user interface), the reasons and the advantages/disadvantages of adopting mechanisms and techniques to fulfill the required tool functionality, and the experiences gained in the implementation.

- *Chapter 5*: A Case Study – illustrates the features of the tool by carrying out a case study. Some of the design and implementation details are further discussed here.

- *Chapter 6*: Evaluation – evaluates the prototype tool by the Cognitive Dimensions framework [Green et al 96] and a user survey.

- *Chapter 7*: Conclusion – summarizes the contributions made by this thesis, and outlines the future work in this research field.

1.6 Summary

In this chapter, we have stated our research topic, the motivation for us to carry out this research, the goals we want to achieve, and the approach we take. We have also provided an overview of the structure and contents of this thesis.

CHAPTER 2 - Background Study

2.1 Introduction

This chapter contains a background study prior to our tool development. We divide materials obtained from the study into two parts: *Basic Concepts* and *Related Research* for the convenience of discussion.

The first part of this chapter covers the theoretical concepts our research is based on, the research areas our project is related to, and some materials from other research that we feel are in some levels of 'classic', 'standardized' or 'well recognized/adopted' and are used in our work. The second part introduces other work closely related to our research that has lessons and experiences we can learn something from.

2.2 Basic Concepts

In the previous chapter, we indicated that our research is the development of a prototype E-whiteboard based sketching tool. Therefore, we discuss concepts and issues regarding topics of E-whiteboard and pen-based computing. Also, as the UML has been selected as the software design notation example for our experimental research, we also discuss basic concepts of the UML, conventional UML CASE tools and object-oriented software design in general.

2.2.1 E-whiteboard and Pen-based Computing

History

The success of research in human-computer interaction (HCI) has fundamentally changed the ways of computing. One example is that virtually all software written today employs graphical user interface toolkits and interfaces builders [Myers 98].

Historically user interfaces evolved from Textural (TUI) to Graphical (GUI), and now to Pen-based (PUI) [Apte et al 93]. In a TUI, the input is keyboard-based character streams. The user must know the syntax and semantics of a description language, and the representation is highly indirect. In a GUI, the mouse has shared a large part of the role for the user input from the keyboard. Through mouse clicks and drag-and-drops, the user gets direct feedback of the visual construction so called "What You See Is What You Get" (WYSIWYG) [Apte et al 93].

The research and development of pen-based computing was started in the early 1960's [Myers 98]. Until recently, because of the evolutionary of its devices and related technologies, the pen-based computing system has emerged and started attracting more and more users. The PUI promises to be a real alternative to the keyboard and mouse based one.

Devices

Recently, small screen devices such as PDAs (Personal Digital Assistants), H/PC (Handheld PC) or cellular phones enjoy enormous popularity [Myers 98]. The phenomenal growth and rising demand as well as reliance of users to use computers anywhere at any time have further driven the future of computers towards to mobile and ubiquitous computing. However, several limitations on the size of the screen and memory of above devices have restricted their usage in many areas to date.

The Tablet PC from Microsoft, which is one of the newest pen-based computing devices, has been launched to the market and started to play an important role in those areas that small devices were unable to [Microsoft 03]. However, it is still not a 'one-off' solution to all needs, e.g. it is hard for a group of developers performing collaborative design tasks on a Tablet PC due to the limitation of its size.

Recently, there have been increasing interests in large screen computing devices. The "E-whiteboard" is such a device that has become a popular way of supporting a wide range of activities. These include: meeting support with collaborative document display, review and annotation [Voida et al 02]; education [Berque et al 02];

presentation control and annotation [Apperley et al 02]; and (early) design [Lank et al 02]. The input advantages of conventional whiteboards are partially replicated with E-whiteboard applications, with additional advantages of digital data capture and display, distributed work support and control via sketching/gesture-based interfaces [Apperley et al 02, Berque et al 02, SMART 03].

Hand-writing/drawing Recognition

Much work has focused on the recognition of hand-writings and hand-drawings which is the major issue in pen-based computing. There are mainly two different mechanisms: *Optical Character Recognition (OCR)* and *Gesture recognition.*

- **OCR** – is often referred as "off-line" handwriting recognition, as it recognizes and converts scanned handwriting images into a text file which then may be used for word processing or other forms of data processing. As such, it is mostly used for office automation and information retrieval and is not quite relevant to our research. However, the **Neural-Network**, an important and advanced mechanism adopted in most OCR tools is a potential alternative may be 'borrowed' in this research area. We did not implement this mechanism due to its complexity and the need of learning process, which we felt might be inefficient for our application. However, we still give a brief introduction of it to provide ideas for those who may read this thesis and also for ourselves in doing the future work.

  The original biological term Neural-Network used here refers to a data modeling tool that is able to capture and represent complex input/output relationships [OCHRE 03]. The motivation for the development of Neural-Network technology stemmed from the desire to develop an artificial system that could perform intelligent tasks similar to those performed by the human brain. It resembles the human brain in the ways of a) it acquires knowledge through learning, and b) the gained knowledge is stored within inter-neuron connection strengths known as synaptic weights.

The most common Neural-Network model is the *Multilayer Perceptron (MLP)* [Fig 2.1]. The goal of this type of network is to create a model that “correctly” maps the input to the output using historical data so that the model can then be used to produce the output when the desired output is unknown. The MLP and many other Neural-Networks learn using an algorithm called *Backpropagation*. With backpropagation, the input data is repeatedly presented to the Neural-Network. With each presentation the output of the Neural-Network is compared to the desired output and an error is computed. This error is then fed back (back-propagated) to the Neural-Network and used to adjust the weights such that the error decreases with each iteration and the neural model gets close and close to produce the desired output. This process is known as “training”. More details of its techniques and application examples can be viewed on those articles/books about this topic [OCHRE 03].

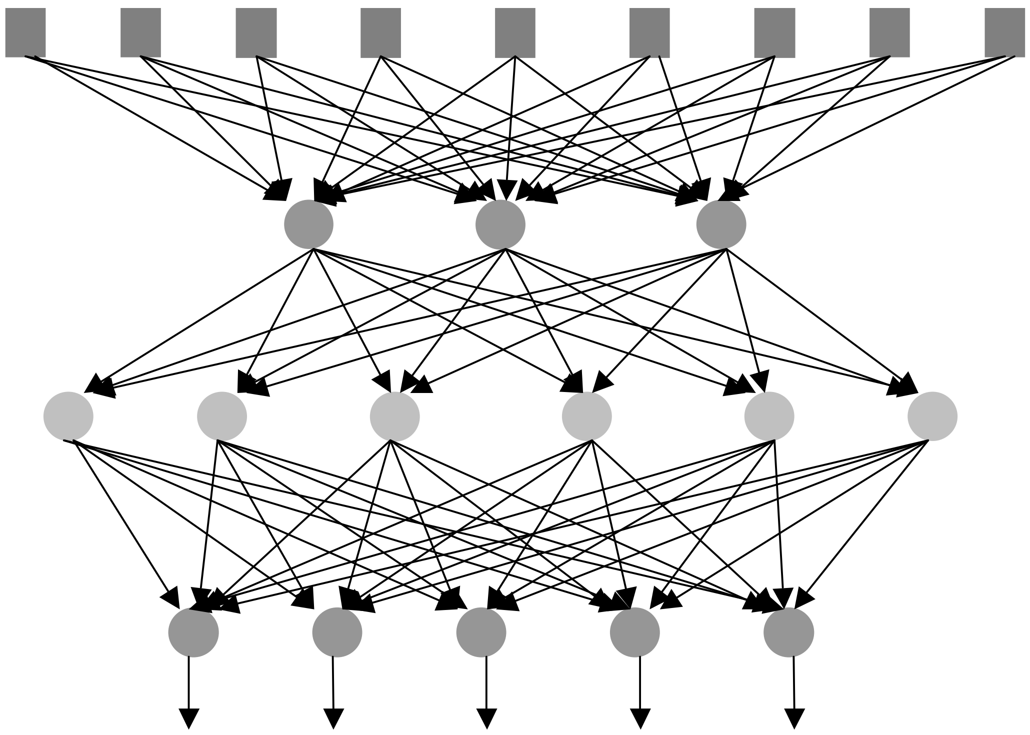


Figure 2.1 Backpropagation Network Architecture

- **Gesture recognition** – as the term used here, refers to deal with the movements performed with one's hand with a pen or stylus. The contents for recognition can be both shapes and texts. Gesture recognition can be seen as the key issue in pen-based computing. Sometimes it is also referred as "on-line" handwriting recognition as the recognition is often performed while the gesture is being drawn.

  Since the first pen-based input device has been invented, it was quite common in those systems to include some gesture recognition, for example in the AMBIT/G system (1968). A gesture-based text editor using proofreading symbols was developed at CMU by Michael Coleman in 1969. Gesture recognition has been used in commercial CAD systems since the 1970s and came to universal notice with the Apple Newton in 1992 [Myers 98]. Teitelman in 1964 developed the first trainable gesture recognizer. A very early demonstration of gesture recognition was Tom Ellis's GRAIL system on the Rand tablet [Myers 98]. More recently, Microsoft's Office XP has facilitated a handwriting-recognition engine which is expected to be used primarily in Tablet PCs for pen-based text input and recognition [Microsoft 03].

  Much research has been carried out on algorithms for gesture recognition. Relatively few efficient algorithms have been published so far. From our research, we found two types of suitable algorithms are available currently: the *Single-Stroke Recognition Algorithm* and the *Multi-Stroke Recognition Algorithm*. We discuss these two types of algorithms in detail below.

  Single-Stroke Recognition Algorithm

  This type of algorithm is single-stroke and feature analysis based, with adopted mechanisms of the pattern matching, or the visual language parsing, or the integration of two [Zhao 93]. Among those proposed algorithms, Rubinee's algorithm [Rubine 96] is the most popular one that has been implemented in many applications.

  In Rubinee's algorithm, a gesture is an array g of P time-stamped sample points:

$$g_p = (x_p, y_p, t_p) \quad 0 <= p < P$$

A vector of features, which are chosen according to the criteria specified in the paper, is extracted from the input gesture.

$$f = [f_1.....f_n]$$

The classic linear discriminator is used for pattern recognition. The recognizer is trained from a number of examples of each gesture.

Gesture class c has weights w_{ci} for $0 <= i < F$, where F is the number of features.

The training problem is to determine the weights w_{ci} from the example gestures. A well known closed formula is used instead of iterative techniques. The formula is known to produce optimal classifiers under certain rather strict normality assumptions on the per-class distributions of feature vectors. The Mahalanobis distance and other methods are used for rejecting ambiguous gestures and outliers.

An input gesture is evaluated by a linear function over the features. The evaluations, v_c, are calculated as follows:

$$V_c = w_{c0} + \sum_{i=1}^{F} w_{ci} f_i \qquad 0 <= c < C$$

The classification of gesture g is the c which maximizes V_c.

The main features of Rubinee's algorithm are that: 1) it combines statistical gesture recognition with direct manipulation techniques; 2) each gesture is recognized independent of its shape and size, but has to be drawn in a single stroke.

Multi-Stroke Recognition Algorithm

This type of algorithm is generally applied to shape recognition, such as diagram components. Most techniques used for the shape recognition in research and

developments are primarily for machine vision and recognition, and are relatively too complex and beyond the requirements in pen-based recognition systems [Zhao 93]. Apte *et al* proposed a 'fast, simple' algorithm for recognizing hand-drawn multi-stroke geometric shapes [Apte et al 93].
Their algorithm is based on two ideas. First, because shapes are geometric entities, they should be recognizable solely by their geometry. No stroke information (i.e. order of input data points, speed of input) is used for analysis; only data points are used. Second, recognition can be achieved by filtering. Each filter categorizes input according to certain criteria. Passing data through filters provides an increasingly refined description of the data.

Three filters are used in their algorithm: 1) *Area-Radio (A_c/A_r) Filter* – which A_c is the area of the convex hull defined by the collected points, and A_r is the area of the coordinate-oriented bounding rectangle. The ratio for rectangles is close to 100%, the ratio for ellipses is approximately 80%, and the ratios for triangles and diamonds are both about 50%. 2) *Triangle-Diamond Filter* – the relation of the left and right corners to the rest of the shape is checked by averaging the Y value of those corners. In a triangle, those corners are near the bottom of the shape. In a diamond, those corners are near the middle of the shape. 3) *P^2/A Filter* – which P is the perimeter and A is the area of the shape. This ratio is a scalar for any shape. For instance, $P^2/A = 16$ for squares of any size. Let $s = width / height$ for the bounding rectangle of a given shape, the P^2/A ratio can be extended to shapes with $s \neq 1$ and can be calculated based on the value s. Though the ratio for lines is theoretically infinite, in practice it is found to be greater than 55.

There are two common difficulties involved in multi-stroke shape recognition. These are segmentation (to identify which strokes belong to which objects) and incomplete specification (shape is not closed – every stroke in a multi-stroke object may not be connected to the other strokes of the object).

The algorithm proposed by Apte *et al* solves the first problem by introducing a proportional time-out technique. After the first stroke, consequent strokes are drawn within a certain time-out period, if the pause is longer than the period, the recognition procedure is triggered and all previous strokes are recognized together as one shape object. The formula they used:

$$time\text{-}out = 500 + 4n\ (ms)$$

where n the number of data points collected is linearly proportional to the number of data points collected so that larger objects will have a longer time-out as the user may have to move the pen some distance to start the next stroke.

To solve the second problem, firstly, the minimum spanning convex hull is derived from the input data points. Then the *Perimeter (P)* and *Area (A)* of the convex hull are computed for use by some of the filters. Each shape employs various filters. Each filter is weighted according to its success in recognizing that shape. For triangles and diamonds, the A_c/A_r and *Triangle-Diamond* filters are used. For circles and lines, only the P^2/A filter is used. And for rectangles and ellipses, both A_c/A_r and P^2/A ratio filters are used. The weighted outputs are combined to select the final shape.

The main features of this algorithm are: 1) it provides a natural way for users drawing geometric objects in multi-strokes; 2) the user needs to pause their drawings to allow the time-out to trigger recognition; 3) objects are recognized independent of its size and number of strokes, but are orientation dependent; 4) it is only used for recognizing geometric shapes.

In the above, we discussed a brief history and the basic concepts about pen-based computing, E-whiteboard, and mechanisms and algorithms presented in other research. Next, we will briefly introduce concepts of UML, CASE tool and software design and their relationships.

2.2.2 UML, CASE Tool and Object-Oriented Software Design

Methodologies for software design have evolved over past decades [Rumbaugh et al 91]. Now, ***Object-Oriented*** design is the most widely adopted one in this area. It is a mapping of real-world phenomena and the concepts of a problem domain to objects, classes, and their relationships in a solution domain. The process of doing so is called modeling. Even though the reality of object-oriented design is much more complex, modeling enjoys a central position in object-orientation [Rumbaugh et al 91]. The fundamental concept in object-oriented modeling is the object. An object contains both data and methods that operate on the data. Objects are said to encapsulate their data by separating their external aspects with which other objects can interact, from their internal details such as their data and implementation details. An object's data can therefore only be accessed via its methods and accessible attributes. Objects are typically composed of smaller objects. Another concept in object-oriented modeling is the *class*. A class is an abstraction of a set of objects with the same behavior (represented as their method) and/or structure (represented as their data). The concepts of class and object are closely related. A single object is an instance of a class. A class defines attributes, pieces of data that instances of the class will have, and methods, the operations on the data. Classes can inherit (all the attributes and methods) from other classes, and also override some of the methods after inherited to provide new implementations. Above is a brief introduction of object-oriented modeling in software design. Detailed information can be found in [Rumbaugh et al 91].

The ***UML*** is the predominant industry standard for object-oriented modeling [UML 03]. It is a large and comprehensive visual modeling language containing a certain set of diagrams used for different modeling purposes. They are Use Case diagram, Class diagram, Package diagram, Collaborate diagram, Sequence diagram, Deploy diagram, Component diagram and State diagram. Each diagram is used to describe a different aspect of the underlying object model being created. For example, Class Diagrams describe the static structure of an object model by presenting their properties, methods and attributes, and inheritance and association relationships between them. Sequence or Collaboration Diagrams describe the dynamic behaviors in the object model by showing the series of messages that get passed between objects or the method-calls that occur in a specific collaboration. Different notations are used in each diagram to map graphical syntax onto the specifying object models that have an exact semantic meaning. Examples of UML diagrams will be presented in next chapter. More detailed explanations of the UML can be found in [Fowler 97].

CASE Tools have been widely adopted among software engineering communities due to its provision of efficiency in supporting the software development. A variety of CASE tools have been created to support the developer's work throughout the software development process [Damm et al 00]. UML tools [UML Tools 03] are examples of them, which allow developers to draw different types of diagrams when they carry out software analysis and design modeling processes. They are commonly installed on computers and take inputs from the keyboard/mouse and display outputs on the screen. However, many studies regarding the topic of "Human-Computer Interaction" have pointed out that developers prefer sketching designs by handwriting rather than doing it using the keyboard/mouse on the computer screen, especially in the early stages of the software design [Damm et al 00]. This may be because that handwriting is more natural for humans to perform creative designs as it is quick and easy to express ideas with more freedom. Another issue is that in the case of collaborative work among developers, which is often required in software design, a computer screen with common size is not suitable as the communication tool. Much research has focused on using the E-whiteboard technologies to solve the above issues.

2.3 Related Research

A number of related research projects are summarized here. These address areas of sketching-based design and sketching-based UML design tools.

SILK [Landay 96]

In their early research on the topic of pen-based computing, Landay and Myers proposed the Garnet and Amulet toolkits and the training tool for the recognizer - Agate. Based on this work, they finally developed a more sophisticated tool called SILK that allows software designers to sketch an interface using an electronic pad and stylus. SILK tries to recognize user interface widgets and other interface elements as soon as they are drawn, though it is not intrusive ('sketchy' look of the drawn components are retained) and users will only be made aware of the results when they choose to exercise the widgets. The recognition uses Rubine's single stroke gesture recognition algorithm. When the designer is satisfied with the early prototype, SILK can transform the sketches into real widgets and graphical objects that can take on a specified 'look and feel' of a standard graphical user interface. At each stage of the process, the designer can switch the sketch into the run mode to test the interface. Testing is done by manipulating it with the mouse, keyboard, or stylus, while SILK stores the history of all drawings for later use. Annotation and editing are also supported in the tool. By the time of the publishing of their paper, the prototype of SILK ran under Common Lisp on both UNIX workstations and an Apple Macintosh with Wacom tablet attached. It supported recognition and operation of several standard widgets, and transformed sketches to an interface with a Motif look-and-feel. It could only recognize a few ways of drawing each widget, and did not support the specification of its behaviors.

We feel that some approaches introduced in SILK are highly useful. These include the retaining of sketched user interface components; the provision of a standard graphical form of recognized sketches; the support of annotation and editing.

However, SILK is not designed to be used on E-whiteboards and thus it does not address the issues relating to the use of the E-whiteboard technologies. Also it is designed to work with the user interface design, the implementation ideas and mechanisms can not be generalized to UML design. Another drawback is the single-stroke recognition algorithm they used for the recognition has the disadvantages which have been discussed in the previous 'Gesture recognition' section of this chapter.

FreeForm [Perlimmer et al 02]

The FreeForm software was developed by Plimmer *et al* as a Visual Basic (VB) Add-In for the design of VB forms. It adopted the same metaphor of the SILK, but utilizes an electronic whiteboard and pen input to support hand-drawn sketching of Visual Basic user interface screens. There are five major parts to the software; the sketch space, storyboard, run mode, recognition engine and conversion of the sketch to a VB form. The sketch space allows the user to draw multiple sketches each depicting a different form, while the storyboard shows miniature views of all the form sketches and allows the user to add links between forms. In the run mode the sketch is shown but can not be altered. The user can navigate between the forms by touching 'hotspots'. The recognition of shapes and characters are also based on Rubine's algorithm with shape/letter library and rule mapping techniques.

We feel that FreeForm can almost be seen as a refined E-whiteboard version of SILK, and thus inherited most advantages and disadvantages from SILK. However, issues of using the E-whiteboard and pen-based input, especially the usability issue, have been addressed in some degree which are useful to our research (please refer to their paper for details).

Knight [Damm et al 00]

The project of developing the Knight tool was undertaken by Damm *et al* at the University of Aarhus, Denmark. It seems to be the work closest to our research. Knight supports collaborative UML modeling using gestures on an electronic whiteboard with pen input. To achieve intuitive interaction, the Knight uses compound gestures and eager recognition. Compound gestures combine gestures that are either close in time or space to one drawing element. Eager recognition, again based on Rubine's algorithm, tries to classify gestures (shapes) while they are being drawn. Text input is supported in five ways: normal keyboard, on-screen Virtual Keyboard, Stylus-based Gestures (as on PDA's), Cirrin, and Quikwrite. Apart from the Stylus-based Gestures, none of others are gesture recognition based. To support the informality, it allows incomplete elements being recognized at a later time; and also, a separate "freehand" mode can be activated in which allowing the user to make arbitrary sketches and annotations. However, the association between a "freehand" element and a "formal" (i.e. recognized) element is not implemented.

We feel that Knight can be seen as a UML version of FreeForm. However, the main difference of Knight is it converts UML sketches into computer drawn diagrams as soon as they are recognized. This does not support the 'look and feel' of conventional whiteboard and thus is seen as its major weakness which limits the exploratory and creative design when working with it. Because of the complexity of the UML diagrams and because of the single-stroke recognition algorithm they used is not sophisticated enough to handle them, Knight only support a small range of UML diagrams which is not seen to be able to generalized to a full functional UML sketching tool or other modeling tools.

Other related research

We briefly discuss some related research applied in domains other than the software design.

Guimbretiere *et al* [Guimbretiere et al 01] developed an "interactive wall" metaphor implementing **FlowMenu**. This is a brainstorming tool uses direct pen-based interaction on a large high resolution display to support both free-hand sketching and displaying high-resolution materials such as 3D models and GUI application windows. Their demonstration of the feasibility and appeal of interacting directly with digital materials on large surfaces is an encouragement to our investigation.

In their paper [Gross et al 96], Gross and Do demonstrated the **Electronic Cocktail Napkin**, a pen-based interface to support intelligent design critiquing, retrieval of relevant items from a case library, and simulation. It uses a simple, trainable-on-the-fly recognition scheme to identify multi-stroke glyphs drawn using a Wacom digitizing tablet, mouse, or Newton PDA. They showed applications of the tool in indexing visual databases for building collection and design and a front end to a local area network design program. Their recognition scheme is said to be able to recognize 'multi-stroke glyphs' and the details of the actual recognition algorithm is not specified in their paper. However, from the description of the scheme in their paper, we can see that it is still a training required, pattern matching based algorithm which is expected to be similar with the single-stroke recognition algorithm we have introduced.

Igarashi *et al* described a tool called **Flatland** [Igarashi et al 00] which intend to support various activities on personal office whiteboard. Their architecture is characterized by its use of freeform strokes as the basic primitive for input and output, flexible screen space segmentation, pluggable applications that can operate on each segment, and build-in history management mechanisms. The dynamic segmentation technique is the major strength of this application.

Chatty and Lecoanet reported [Chatty et al 99] the project, IMAGINE, which represented the second generation of a graphical interface for air traffic control. Their prototype, **GRIGRI**, uses a high-resolution touch screen and provides mark-based input through the screen. They attempted to use gestures and multi-modal techniques to make interaction faster, and closer to the controllers' habits. This challenging application in the high demanding air traffic control field again encouraged us investigating in this pen sketching based research area

Bailey and Konstan [Bailey et al 03] reported on an evaluation comparing **DEMAIS** to pencil and paper and Authorware for the exploration and communication of behavior in early multimedia design. They claim that DEMAIS is an informal design tool that helps a multimedia designer explore and communicate temporal and interactive (behavioral) design ideas better than existing tools. DEMAIS provides storyboard, voice script, and multi-view editors. There are no recognitions in DEMAIS. Instead, when a designer sketches an ink stroke that connects rectangular strokes, text objects, or imported media objects, DEMAIS interprets that stroke as a behavioral stroke and transforms it into a low-fidelity, functional prototype. All sketches are remained on the storyboard and can be modified later. Again the approach of remaining the sketches supports our concept of how an E-whiteboard based sketching tool should act like.

2.4 Summary

In this chapter, we have summarized the material gained from our background study. Basic concepts and related research have been discussed to some degree.

We feel that some approaches adopted in other research are not good choices to be implemented in our tool. For example, the single-stroke recognition algorithms implemented in most current applications may not be sufficient enough for our application due to its unnatural way of drawing and the need of extra training processes. The approach of immediate conversion of sketches into a formalized form as soon as they are recognized is not a nature behavior that an E-whiteboard sketching tool shall have.

On the other hand, we feel that the multi-stroke algorithm can be a good alternative to our tool, though more experiments and extra works may be required to make it more efficient and powerful. We were also influenced by some of other applications e.g. Plimmer's observations in her Freeform work that retaining a sketch form encourages more experimentation with design. Therefore, our approach in developing a new E-whiteboard based UML sketching tool, is to explore the retention of the hand-drawn sketch "look and feel" of real whiteboards with UML sketches, while retaining the ability to recognize and convert the sketches to more formal diagrams.

CHAPTER 3 - Requirements Specification

3.1 Introduction

The requirements of this project came from three sources: our goal, background study, and a survey. The former twos are quite usual to research-type applications. However, it is not our goal to build a tool just in our own interests and in the way of how we 'feel' it should be. We intend to make our tool to be able to meet commercial standards of possible future developments, and as such, the tool has to be made in the way of how potential users would feel it should be. It is always crucial for the success of any type of software development to get users involved as early as possible. Therefore, a survey was carried out to help us analyzing the requirements of this application.

In this chapter, we determine the requirements by analyzing the basic functionality the tool will provide; the UML semantic definition and constraints the tool will follow; and the ideas/suggestions summarized from the survey we have carried out. Then, we present a conceptual object model we developed for the tool based those requirements. The UML *Class Diagram* is used to illustrate the *Basic Object Structure* (static view) of the tool, and the *Sequence Diagram* is used to represent the *Dynamic Behaviors* (dynamic view) of the tool. From our previous study, we can see that there are many possible ways of implementing this tool. We intend to keep the model as simple and abstractive as possible at this early stage to avoid restricting ourselves from many potential choices in later design and implementation stages.

3.2 Basic Tool Functionality

From our goal and the background study, we summarize the basic functionality required by the tool below:

- The tool will enable users to use a pen-based electronic whiteboard system to sketch UML diagrams and any other free hand writings/sketches in a similar way of drawing them on a conventional whiteboard. Those sketches will be captured and displayed on the E-whiteboard as long as the users desire to provide the retention of the hand-drawn "look and feel" of real whiteboards. The users will be able to use pen-based input to manipulate sketches (e.g. to move, copy, replace, delete sketches) which provides advantages of the tool over conventional whiteboards.

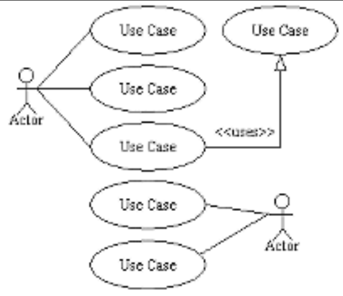
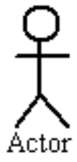
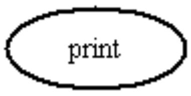
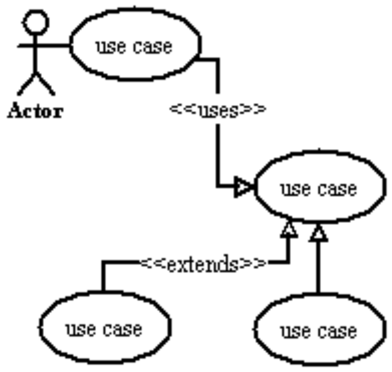
- The tool will be able to progressively recognize sketched UML diagrams together with necessary identifying/specifying text. It will also be able to represent them in a formalized view when desired to provide feedbacks on semantic constraints. The users will be informed by a proper manner whether a sketched item has been recognized or not.

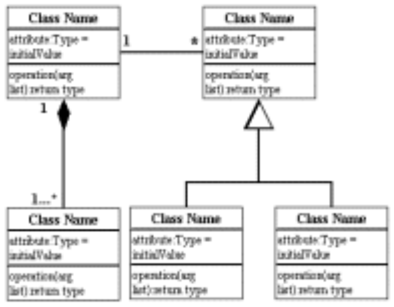
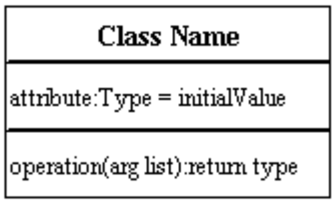
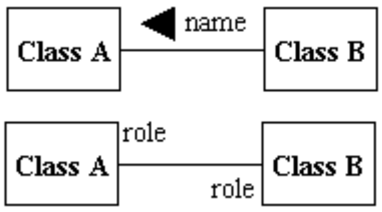
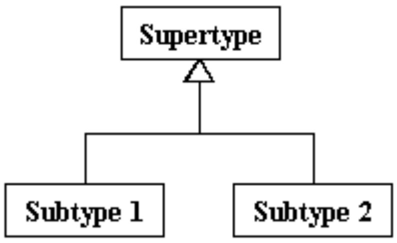
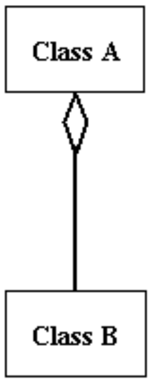
- The tool will be able to save sketches as a re-loadable file in a physical storage or export formalized diagrams to a standard UML CASE tool for the further work.

3.3 UML Semantics

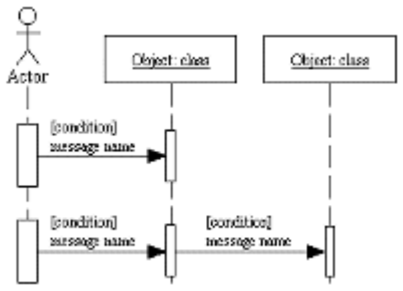
We have chosen to experiment with UML notations in our research for the reasons of: 1) The UML has become a de-facto standard visual modeling language used in the software analysis and design. It is well understood by us and most software designers, and thus it may be easer for us to implement it in our tool as well as to perform the usability evaluation by the users which will be software designers. 2) The richness and complexity of the notations contained in different types of UML diagrams challenges us to develop a more sophisticated and extensive tool. More experiences will be gained from the experiment.

We did not implement all types of UML diagrams due to time restrictions and the fact that the main purpose of this project is to investigate the interest issues in this area rather than to produce a commercialized tool supporting the whole range of UML diagram types. In the reality, it does not always happen that all UML diagrams are used in every software development. Some diagrams are more frequently used e.g. the Use Case diagram, Class diagram, Sequence diagram, etc.; some are less frequently used e.g. the Component diagram, Package diagram, etc.; and some may never been used by some developers e.g. the Activity diagram. And yet many conventional UML CASE tools do not support all UML diagram types [UML Tools 03]. We avoided experimenting with similar diagram types e.g. the Object diagram is very close to the Class diagram (in this case, the Class diagram is chosen to be implemented in our tool as it is more frequently used then the Object diagram). We also avoided implementing some diagram typess which are not often used e.g. the Activity/State diagram. Table 3.1 below illustrates the semantics definitions and constraints [UML Center 03] of the *five* types of UML diagrams (out of the total *nine* UML diagram types) supported by our tool:

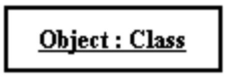
Use Case Diagram	
 Use Case Use Case Use Case Use Case Actor <<uses>> Use Case Use Case Actor	Use case diagrams model the functionality of system using actors and use cases. Use cases are services or functions provided by the system to its users.
	**Actor** An actor is the user of a system. It is represented as a stick figure with a label of its name.
  use case Actor <<uses>> use case <<extends>> use case use case	**Use Case** The use case is draw using an oval with labeled name that represent a system's function. **Relationships** Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.

Class Diagram	
 Class Name attribute:Type = initialValue operation(arg list) return type 1 * 1 1...*	Class diagrams are the most important UML diagram for object oriented modeling. They describe the static structure of a system.
 Class Name attribute:Type = initialValue operation(arg list):return type	**Class** A class is illustrated with a rectangle divided into compartments. The name of the class is placed in the first partition (centered, bolded, and capitalized), the attributes are listed in the second partition, and the operations are listed in the third.
 Class A name Class B Class A role role Class B	**Association** An association represents a static relationship between classes. Its name is placed above, on, or below the association line. A filled arrow is used to indicate the direction of the relationship. Its role names are placed near two ends of an association. Roles represent the way the two classes see each other.
	**Generalization** Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another.
	**Aggregation** Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond end in an aggregation relationship points toward the "whole" class or the aggregate.

Sequence Diagram

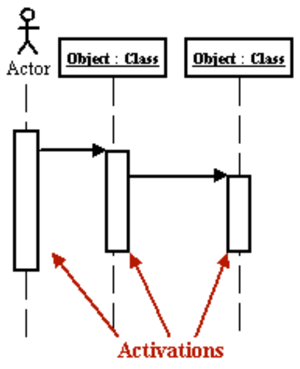


Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.



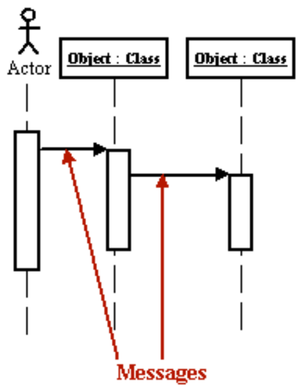
Class role

A class role describes the way an object will behave in context. The UML object symbol with out its attributes is used to illustrate a class role.



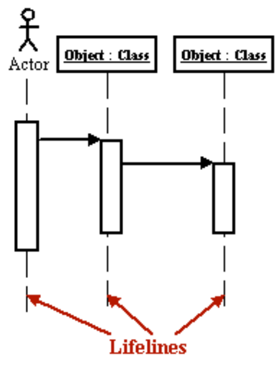
Activation

An activation box represents the time an object needs to complete a task.



Message

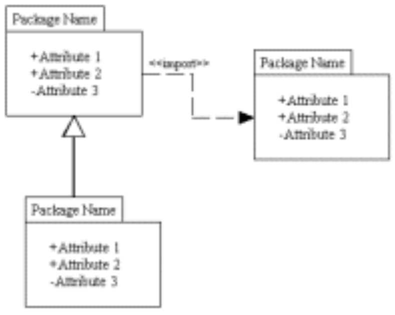
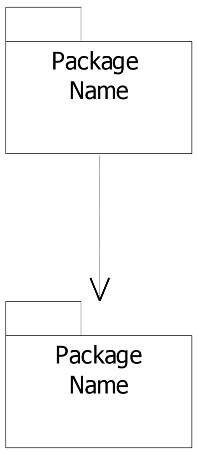
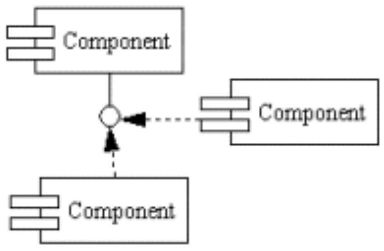
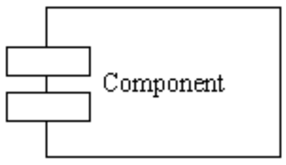
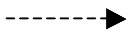
A message is an arrow that represents the communication between objects.



Lifeline

A lifeline is a vertical dashed line that indicates the object's presence over time.

<<import>>

Package Diagram	
 Package Name +Attribute 1 +Attribute 2 -Attribute 3 <<import>>	Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique. Package diagrams organize elements of a system into related groups to minimize dependencies between packages.
	**Package** A package is illustrated using a tabbed folder. Its name is written on the tab or inside the folder. **Dependency** Dependency defines a relationship in which changes to one package will affect another package. It is represented as an dashed arrow between two packages.
Component Diagram	
	Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.
	**Component** A component is a physical building block of the system. It is represented as a rectangle with two tabs.
	Interface An interface describes a group of operations used or created by components.
	**Dependency** A relationship similar with the one in the package diagram.

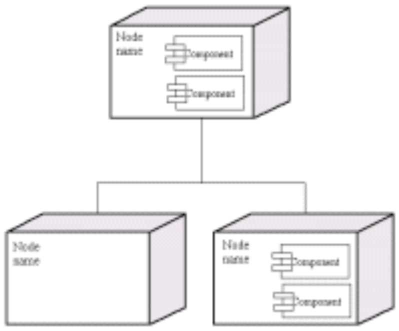
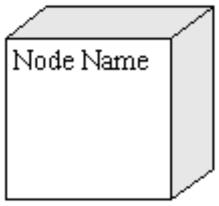
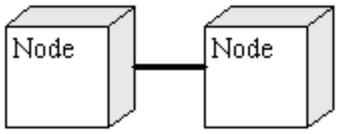
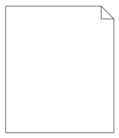
Deployment Diagram	
 Node name Component Component Node name Node name Component Component	Deployment diagrams depict the physical resources in a system, including nodes, components, and connections.
	**Node** A node is a physical resource that executes code components.
	**Association** Association refers to a physical connection between nodes, such as Ethernet.
Note	
	A paper-like icon represents the note symbol which is a place for writing additional descriptions, explanations or other extra information of those notations listed above.

Table 3.1 UML diagrams semantics definitions and constraints

The above UML semantics definitions and constraints will be further taken into account in the design and implementation of the tool development discussed in next chapter.

3.4 Survey

The survey was carried out among a small number of software designers with experience using UML and the whiteboard for software design. It was a fairly informal one which sometimes was rather a 'brainstorming' among participants including ourselves. We felt that this kind of survey had more 'free spaces' for us to capture useful ideas. A typical process of the survey was: 1) first a brief introduction on the research we were doing was introduced. 2) After that the designers were asked to perform an UML analysis/design task on a whiteboard for a real case taken from their own projects. If they did not have any suitable one, the case introduced in chapter 5 (‘Case Study’) was used. 3) While they were sketching the UML designs on the whiteboard, we observed and took notes on how they used a whiteboard for collaboratively modeling software using UML. Open-ending questions were also asked like:

- What do you think are important issues in this research?
- What functionality should the tool provide?
- In what ways should our tool differ compared with a conventional whiteboard?
- In what ways should our tool differ compared with a conventional UML CASE tool?
- What do you expect the tool will look like?
- How do you think the tool will be used?
- What ideas have you got for implementing the tool?

These questions were really examples of the starting points of conversations in the discussion. Substantial questions were followed accordingly, and the discussions and their answers were documented.

The result of the survey has been analyzed and summarized below. However, as this is not a commercial development, we only list those we feel are crucial and also feasible to us to build a successful tool in our research.

- It should be very close of the ways for a user sketching UML on the E-whiteboard and on a conventional whiteboard. However, the E-whiteboard based sketching tool should provide extra features then a conventional whiteboard.

- The users should get as less interruptions as possible while sketching, while they still expect feedbacks of whether the recognitions to the sketches are successful and whether they have been correctly recognized (have a formalized view to check if they meet the UML semantics constraints).

- The ability of manipulating sketches (moving, copying, replacing, deleting of drawn sketches) easily and efficiently should be supported to the design work, and it is preferred they are pen-gesture based rather than other manner based e.g. clicking on modality buttons. In fact, it is expected we implement as many tool functions as possible based on pen-gesture then other methods, e.g. the on-screen keyboard or similar mechanisms for entering text are not ideal. This is seen as one of the main differences between our tool and the conventional UML CASE tools.

- It is observed that although the whiteboard is usually used for collaborative team work, cases of more than one pen drawing on the surface of a whiteboard simultaneously are very rare. Therefore, we do not see it is important in this project to implement mechanisms for enabling the tool handling simultaneous inputs from more than one pen.

- It is observed that users often use as much space of the whiteboard as possible, and they may stand at any site of the whiteboard while working. Different colors and line style/width are used in the sketching. These issues need to considered when design the user interface.
- It is important for the tool to have a good performance on the reaction speed and the recognition accuracy to increase the quality and usability of the tool.

3.5 Summary of Detailed Requirements

Based on the above analysis, we further defined more detailed requirements to be fulfilled in the development of the tool. We summarize them into six categories: *Core Functionality, UML Implementation, Hardware, Usability, Performance, and Extensibility*. The first one is really an alternative to the Use Case diagram which is usually used in this context for codifying the requirements for systems. However, we use our own diagram and description here which we feel are clearer then the Use Case diagram for representing high level functional requirements of the tool. The rest ones can be seen as a non-functional specification of the tool requirements.

Core functionality

We visualize the core functionality of the tool in Figure 3.1.

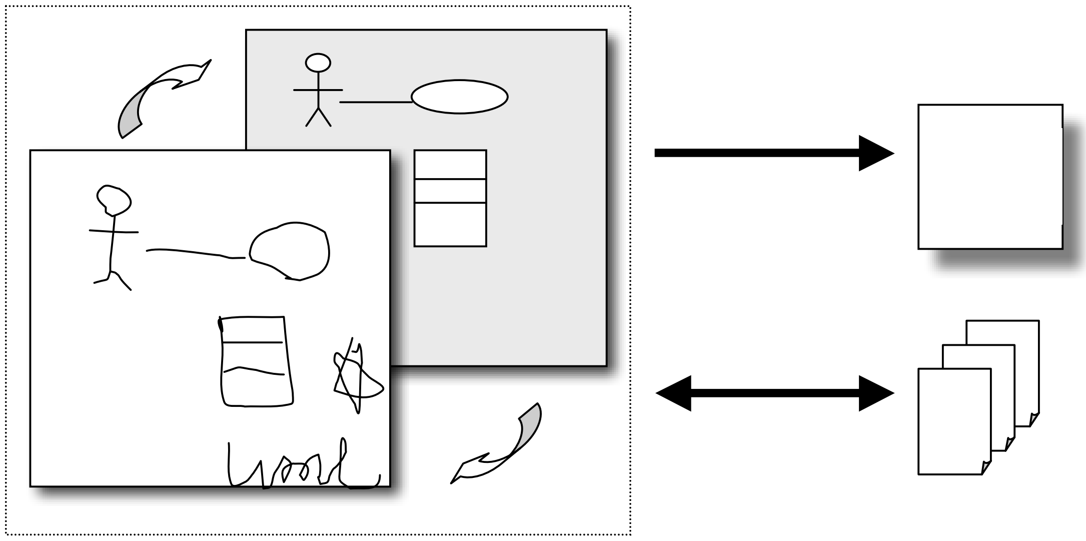


Figure 3.1: Visualization of Core Functionality of the Tool

There are two views in the tool: ***Sketch View*** and ***Formal View***. The ***Sketch View*** is the main working place for users sketching UML diagrams as well as any other secondary notations using an electronic pen with different colors and line widths. The sketches in this view are retained as long as users desired. Recognitions of those sketches are progressively done upon a UML construct or a secondary notation or any item has been drawn. Users will be informed by a proper manner whether recognition is successful or not. Only recognized UML constructs will be represented in a formalized form in the ***Formal View***. However, the ***Formal View*** will not be seen unless users choose to. Two views can be switched easily to provide different 'look and feel' as well as to allow users checking UML semantics constraints. Sketches can be manipulated (moved, copied, deleted, replaced, etc) via pen-gesture based techniques. Two views are completely integrated, i.e. manipulations to an item in one view will reflect the co-related one in another view. Sketches can be saved into files and may be loaded again for later work. Formalized UML diagrams can be exported to a third party UML CASE tool for future work.

UML Implementation

The implementation of UML design in the tool should follow the UML semantics definitions and constraints summarized in Table 3.1.

Hardware

The hardware refers to the E-whiteboard system used in our tool. The constraint of this issue is we have to use one that is available in current market or developed by other research partners, as we dot have time to build up one ourselves. Most E-whiteboard systems [Apperley et al 02, MIMIO 03, SMART 03] will typically have a whiteboard-like surface, a set of stylus (electronic inkless pens) and a pen movement capturing device (e.g. a ultrasonic capturing bar). The captured signals of the pen movements are then transferred into a connected computer. Normally, an application will run on the computer to process the inputted signals and produce outputs. A projector linked to the computer will project the outputs of the application back to the whiteboard-like surface to simulate the pen movement.

Usability

The requirements regarding the usability of the tool are mostly related to the user interface issues as it is quite unique in the size and also the way it works when sketching on an E-whiteboard. Some key points are listed below:

- The user interface should have as much sketching space as possible.
- The user interface should be properly designed and implemented to enable pen-based input for drawing and manipulating sketches.
- The user interface should have an effective way of providing other basic functions, e.g. changing color or drawing style/width, navigating history, etc.
- Functional controls in the user interface shall be easily reached from any position.

- Rich, but not ‘too much’, and less interruptive feedback/indication manner on recognition results should be well designed and implemented.

Performance

Performance is always a critical fact to the acceptance of the software. The tool should have reasonable speed, i.e. ‘immediate’ capturing and displaying after sketching (< 0.5 second) and fast recognition (< 0.25-0.5 second), and accurate recognition rate (> 50%) to enhance the usability.

Extensibility

One of our ambitions is that following this research, we may further extend our tool to be used not just for supporting the UML design sketching but also to be used in other areas. Extensive architecture and mechanisms should be designed and implemented to the tool to fulfill this goal.

3.6 Basic Object Structure

The basic object structure is presented in Figure 3.2.

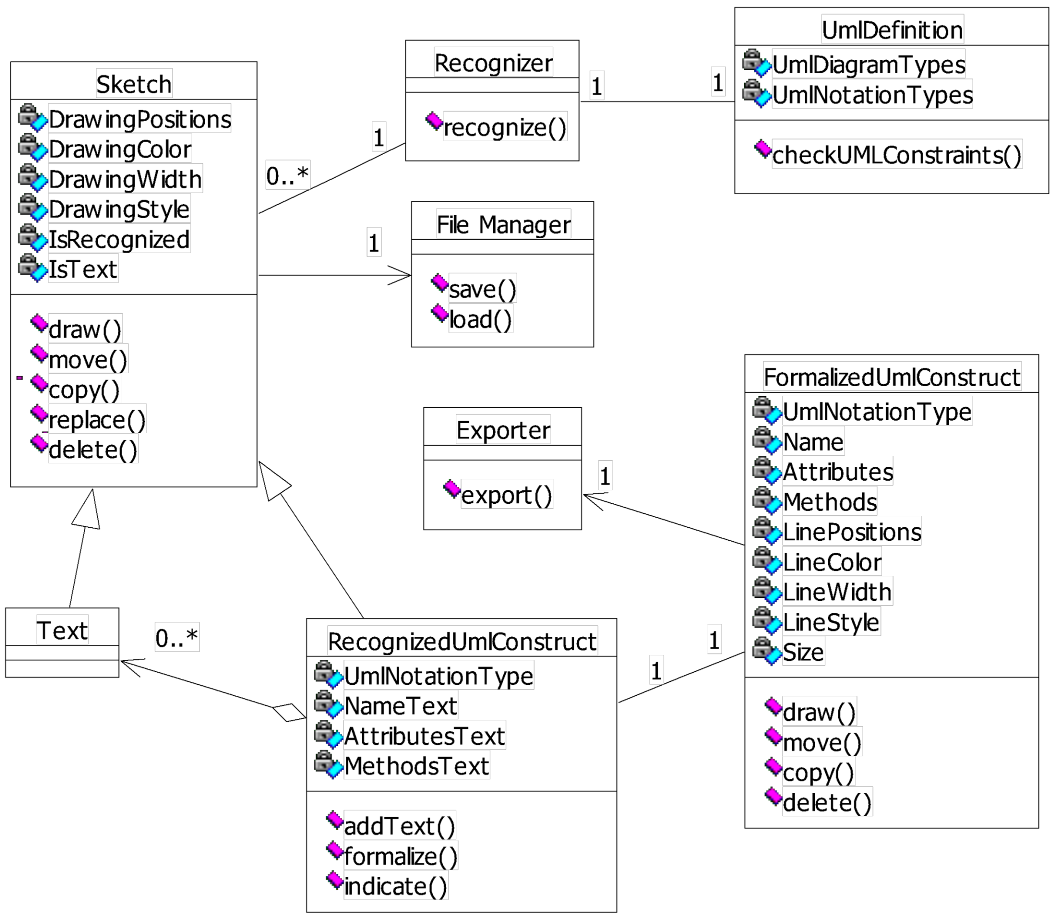


Figure 3.2 Basic Object Structure

The **Sketch** object represents a class of individual sketches drawn by the user. It is the basic unit of drawing items which may be a UML constructor, a secondary notation or anything the user wish to draw on the surface of an E-whiteboard it contains properties of the color, the width (thickness of the pen point) and a style (continuousness – solid or dotted/dashed). It can also be moved, copied, replaced and deleted triggered by pen manipulation events. After a Sketch is drawn, it is passed to the **Recognizer** to be recognized.

The **Recognizer** object carries out the most important function of the tool – recognition. The details about the recognition algorithms/mechanisms adopted by this object will be discussed in-depth in next chapter. Upon the successes of the recognition, it invokes the **UmlDefinition** object to further check if a recognized

UML construct meets the UML semantics constraints, and the final result will be passed back to the Sketch object.

The **UmlDefinition** object contains the information about those UML diagram/notation types supported by the tool and a method for checking if a recognized UML construct meets the UML semantics constraints. If it is the UML construct will be signed to a proper UML diagram/notation type.

The **RecognizedUmlConstruct** object is a special type (generalization) of the Sketch object – a recognized and 'valid' sketch of UML construct. After the object is recognized, the method Indicate is used to inform the user of the success of the recognition, and the Formalize method to formalize into the **FormalizedUmlConstruct** object. Other than all properties inherited from the Sketch object, it can also have properties of the name/attribute(s)/method(s) according to the UML definition. These properties are defined by adding the **Text** object(s) to it.

The **FormalizedUmlConstruct** object represents a computer generated UML construct which is formalized from the RecognizedUmlConstruct object. It contains properties and methods similar to those in the RecognizedUmlConstruct object to enable integration between two objects in different views. For example, if the move() method is called in the RecognizedUmlConstruct, the method with the same name will be triggered to move the FormalizedUmlConstruct object to a new position accordingly, and vice-versa.
The **Text** object is another special type (generalization) of the Sketch object, while also a part of (aggregation) the Recognized UML Constructer object. It inherits all properties and methods from the Sketch object. When the user adds text (name/attribute(s)/method(s)) to the Recognized UML Constructer object, the sketches of the text are recognized by the Recognizer and the results are passed and stored in the Recognized UML Constructer object' properties.

The **File Manager** object is responsible for saving/loading a Sketch or a group of Sketches into/from a file for later works. The **Exporter** object is responsible for exporting the FormalizedUmlConstruct objects to an external UML CASE tool.

3.7 Dynamic Behaviors

We divide the dynamic behaviors of the tool into four categories: 'Sketch and Manipulate', 'Recognize', 'Formalize and Synchronize', and 'Save/Load and Export'. Sequence diagrams are used to illustrate those dynamic behaviors.

Sketch and Manipulate

This part of behaviors deals with the initial setting of the properties of the Sketch object when it is sketched by the user at the first time or resetting its properties after being manipulated, and are illustrated in Figure 3.3. A User object is added to the sequence diagram to represent the user's interaction (sketching and manipulation) with the tool. The manipulations can also be performed to those FormalizedUmlConstruct object by the user. They are shown in Figure 3.5 under the 'Formalize and Synchronize' categories to make the diagrams more readable.

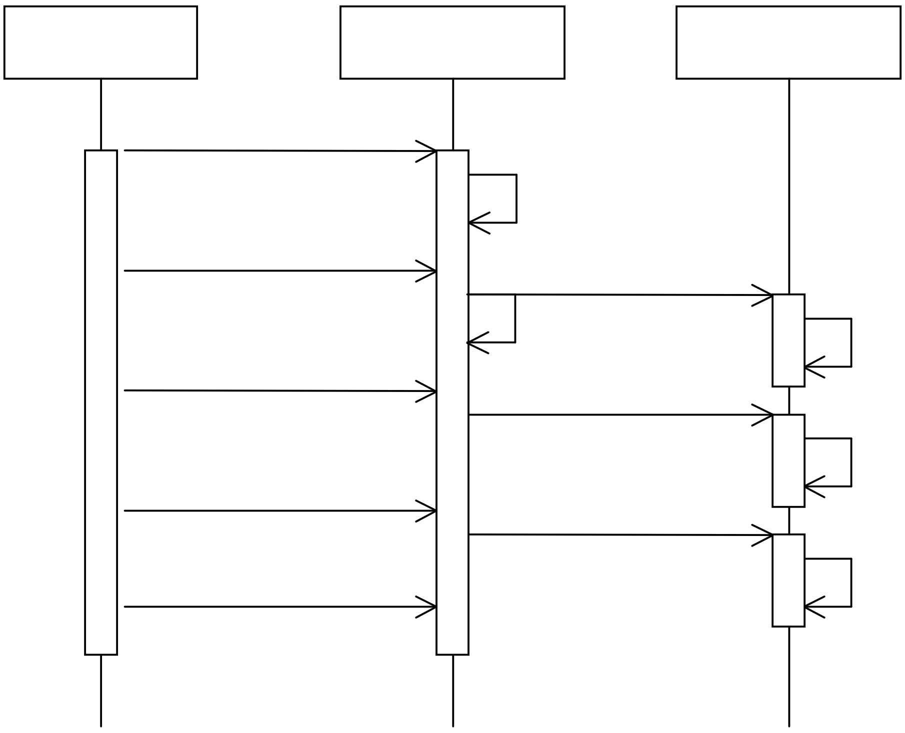


Figure 3.3 Sketch and Manipulate

The Sketch is drawn according to those drawing information (position, color, width etc.) captured from the pen-inputs after it has been sketched by the User. The Sketch's properties are set from the drawing information. When a manipulation occurred, the existing Sketch object is cleaned up from the drawing space in the user interface. Then, if it is a copy() or move(), a new Sketch object is generated and drawn with all same properties except a new position (determined by the pen input). The difference between them is the original Sketch object is re-drawn if the manipulation is a copy(). The replace() manipulation takes a very similar approach to the move() method. The difference of the replace() method compare to the move() method is that the new Sketch object has a close position to the original one but the other properties are very different with those of the original Sketch object. When the delete() manipulation occurs, the Sketch object is not re-drawn, i.e. is 'deleted'.

Recognize

Figure 3.4 illustrates this behavior which involves operations the tool takes when performing the recognition task.

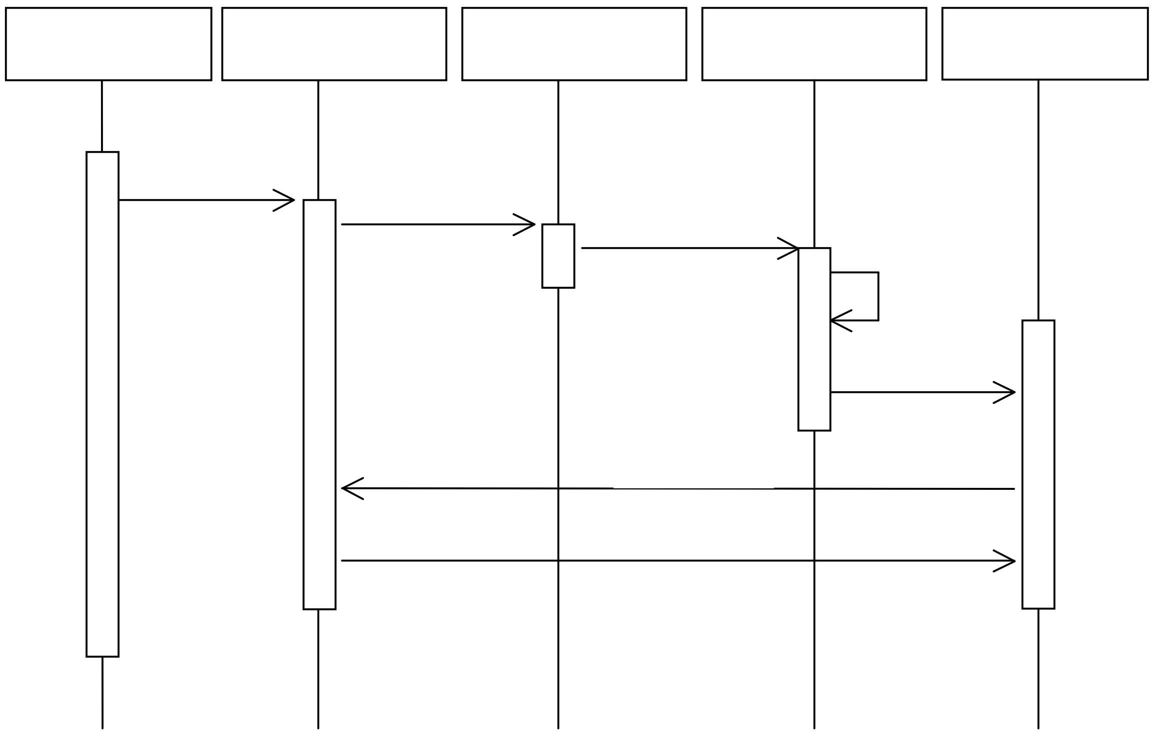


Figure 3.4 Recognize

As soon as a Sketch is drawn by the User (this part is omitted here, please refer to Figure 3.3), the Sketch object is passed to the Recognizer and the recognize() method is invoked to perform the recognition. If the Sketch object is recognized by the Recognizer, it is further passed to the UmlDefinition object for checking the UML semantics constraints, and the UML notation/diagram type will be define if this is successful. The RecognizedUmlConstruct object is then generated from the Sketch object inheriting all properties and methods of the Sketch object. The RecognizedUmlConstruct object carries out its own operations: 1) to indicate() that it has been recognized, 2) to addText() of its name/attributes/methods by invoking the Text object. The Text object, which is also a generalized type of the Sketch object, is then passed to the Recognizer to be recognized. The result of the recognition of the Text object will be returned from the Recognizer. 3) to formalize() itself into a computer drawn FormalizedUmlConstruct object (this will be illustrated in Figure 3.5 under next category).

Formalize and Synchronize

This part is really supplementary to previous two categories which illustrate the formalization behavior and the integration behavior to keep corresponding objects synchronized in two different views – the sketch view and the formal view. (Figure 3.5)

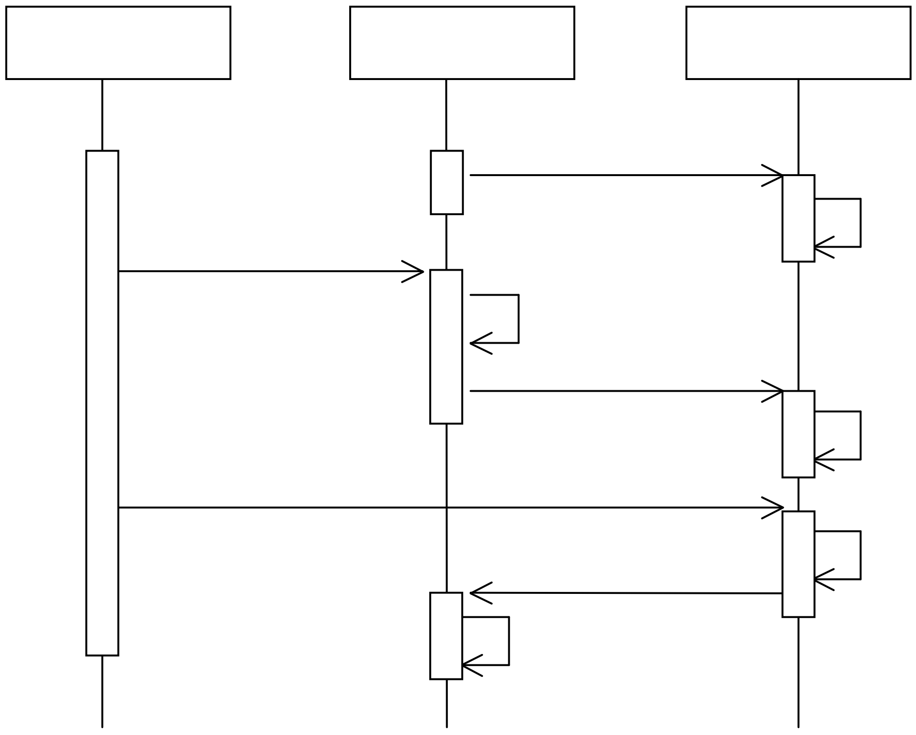


Figure 3.5 Formalize and Synchronize

Upon the success of the recognition (this part is omitted her, please refer to Figure 3.4), the FormalizedUmlConstruct object is generated and drawn by invoking the formalize() method and passing required parameters (position, etc.). The user's manipulations of move(), copy(), replace() and delete() to the RecognizedUmlConstruct object will cause the object being re-drawn, and then the Synchronization information is passed to the FormalizedUmlConstruct object and the FormalizedUmlConstruct object is re-drawn accordingly. The user only can move(), copy() and delete() the FormalizedUmlConstruct object but not the replace (no sketching is done in the formal view). The consequent operations are exact the same as to those discussed above.

Save/Load and Export

There are only a single save/load() operation invoked by the Sketch object and performed by the FileManager object, and a single export() operation invoked by the FormalizedUmlConstruct object and performed by the Exporter object. These two behaviors are fairly straightforward, and thus we omit their Sequence diagrams here but rather leave the underlying technique for implementing them to be discussed further in next chapter.

3.8 Summary

In this chapter, we analyzed and defined the requirements specification for our tool. A conceptual object model has been established representing the basic object structure and dynamic behaviors of the tool. This chapter is then the 'blue-print' for further design and implementation of the tool development.

CHAPTER 4 - Design and Implementation

4.1 Introduction

After defining the requirements of our tool, we now discuss its design and implementation. In this prototype application, although decisions on the design and implementation choices have been carefully made to best meet the specification, some of them are still not sophisticated enough from the perspective of a commercialized development. However, this study do prove the validity of our concept in developing a large e-whiteboard based UML sketching tool, as well as the usefulness of the design ideas and implementation experiences to other efforts in this research area. Discussions on the advantages and disadvantages of the design and implementation choices we have made are provided.

In this chapter, we first present our architecture of the tool. The implementation details of each 'building block' in the architecture are also discussed. Then we present the recognition algorithms used in our tool. The reason for this issue to be discussed separately is because that it is the most important and complex part of the application which influences other design and implementation choices. Next is the presentation of mechanisms adopted to achieve the functions required by the tool. A detailed object model is represented to help illustrating those mechanisms. After that, we discuss issues of the user interface design and implementation which have special considerations due to the use of the unusual devices. Finally, we provide the experiences we have gained from the tool implementation.

4.2 Architecture

Figure 4.1 illustrates the architecture designed to fulfill the functionality required by the tool. It consists of: 1) the E-whiteboard and its input/output, 2) software components (represented as grey cubes in the figure), 3) data storages (files/database), and 4) internal/external communications (represented as arrows in the figure). We explain this architecture and discuss its implementation details below.

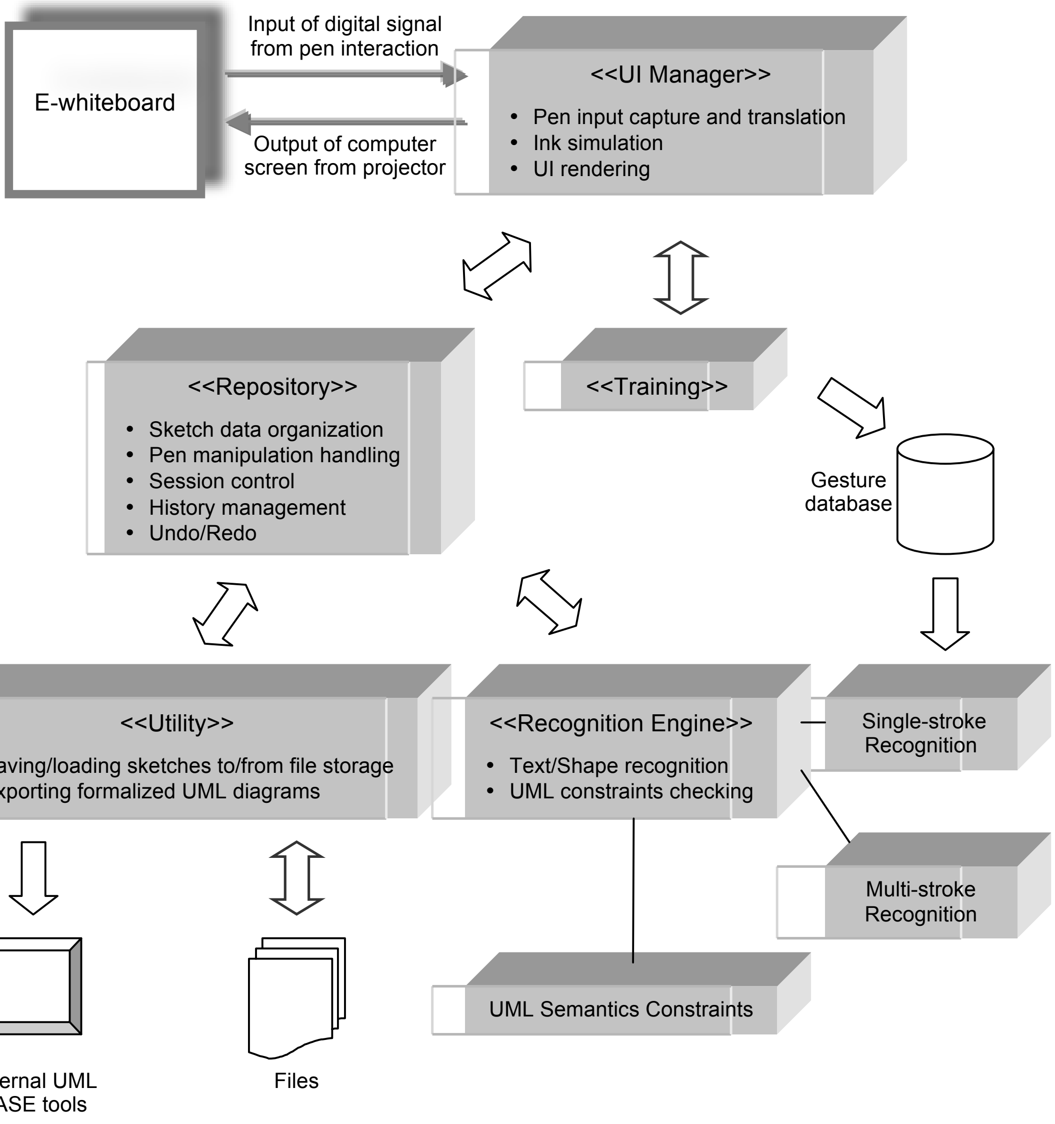
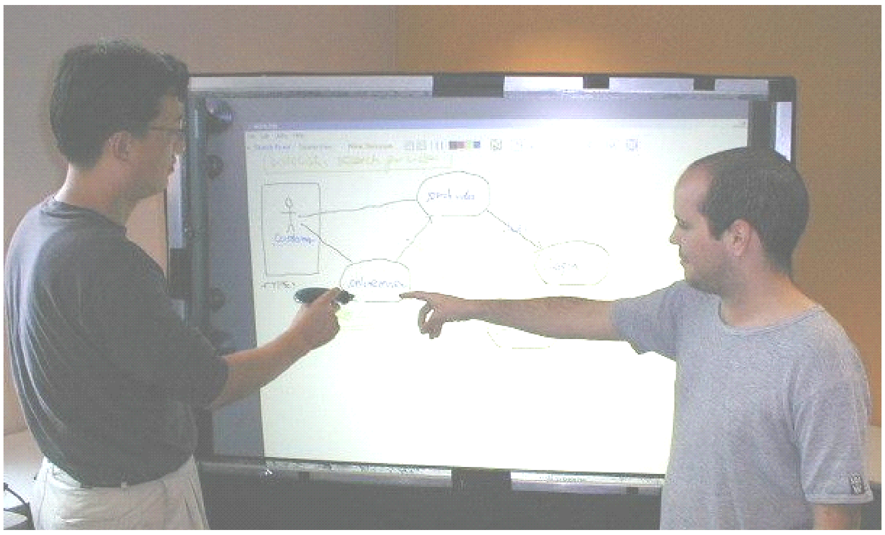

E-whiteboard
Input of digital signal from pen interaction
Output of computer screen from projector
<<UI Manager>>
• Pen input capture and translation
• Ink simulation
• UI rendering
<<Repository>>
• Sketch data organization
• Pen manipulation handling
• Session control
• History management
• Undo/Redo
<<Training>>
Gesture database
<<Utility>>
• Saving/loading sketches to/from file storage
• Exporting formalized UML diagrams
<<Recognition Engine>>
• Text/Shape recognition
• UML constraints checking
Single-stroke Recognition
Multi-stroke Recognition
UML Semantics Constraints
External UML CASE tools
Files

Figure 4.1 Tool architecture

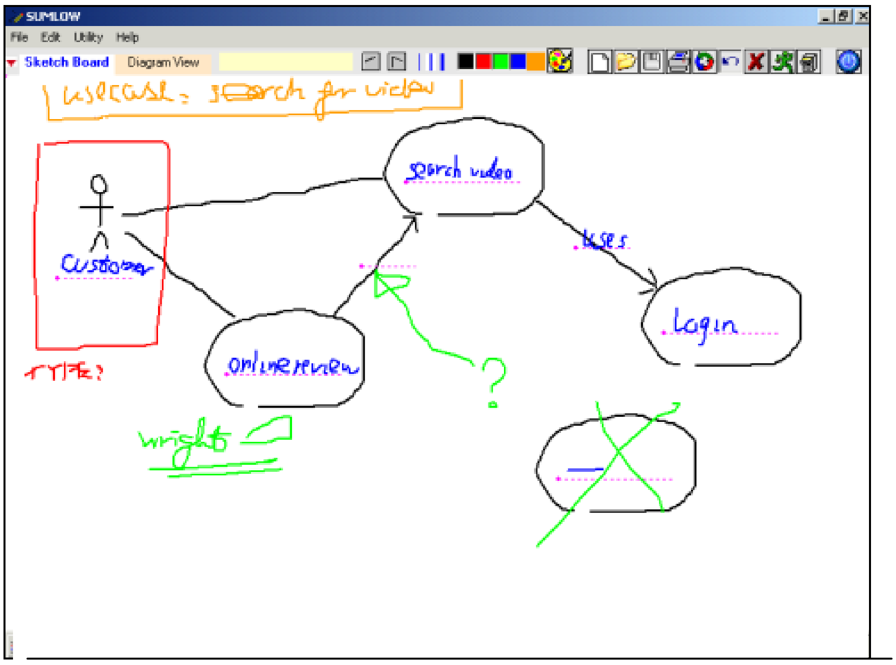
The E-whiteboard System

Figure 4.2 shows our tool in use: a) Designers stand around the e-whiteboard, while one of them is using a stylus to sketch UML diagrams on the surface of the e-whiteboard. The vertical bar attached on the left side of the e-whiteboard is the device which captures the pen inputs. Other devices are contained inside the black box. b)

The sketched UML use case diagram (this example will be discussed in next chapter – 'Case Study').



(a)



(b)

Figure 4.2 (a) Designers around the e-whiteboard; (b) sketched UML use case diagram on the surface

The above figure can be compared with Figure 1.1 in the first chapter of the thesis. We can see that the 'look and feel' of the conventional whiteboard and the e-whiteboard appears very much close. However, the underlying techniques and supported functions have made the e-whiteboard to be very different with the conventional whiteboard. We will further discuss the differences in this and later chapters.

Figure 4.3 illustrates the hardware structure and the input/output data flow of the E-whiteboard system. The reason of choosing the particular products used in the system is just due to their more immediate availabilities at the time of doing this research. It is very possible that other similar products have higher qualities and better performances than those have been used in our tool. However, it is not our purpose to compare the quality and performance of different alternative products. In fact, we did find that users are not very satisfied with some devices in the usability evaluation. However, we do not see this as a significant problem at this prototyping stage as it can be solved by using better products when it is required.

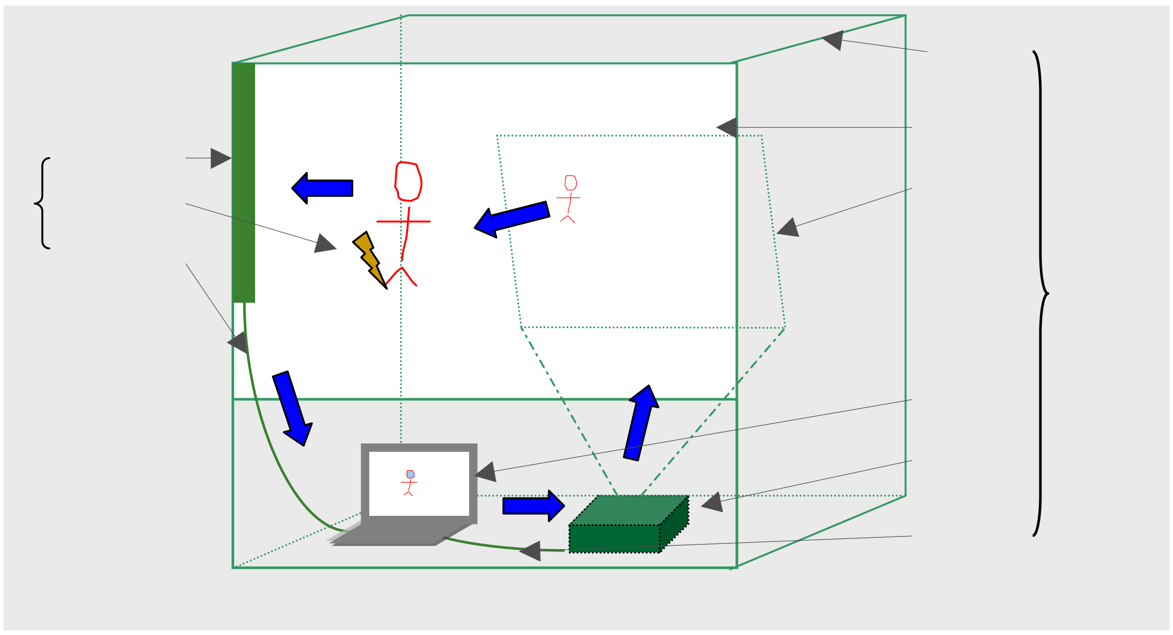


Figure 4.3 Hardware structure and the input/output data flow of the E-whiteboard

From the above figure, we can see that there are two major parts in the E-whiteboard system: the **MIMIO®** [MIMIO 03] and the **LIDS** (Large Image Display Surface) [Apperley et al 02]. The blue block arrows represent input/output data flows between each device of them.

- **MIMIO** – is a tool which provides the feedback of what the user draws on a display surface. It consists of a set of devices and supporting software. The devices used in our tool are: a capture bar, an electronic pen (inkless stylus) and a connection cable. The capture bar contains a sensor which captures ultrasonic signals emitted from the stylus when it is pressed against and moved on a surface. The signals provide the sensor a means of the positions of the stylus with respect to the drawing area, and are sent as digital data to a computer via a connected cable for processing.

  According to the specifications of the provided MIMIO software, the minimum system requirements of the computer should be the following:
  - Personal Computer: IBM compatible 100 MHz
  - Available Serial Port (DB9) and PS/2 keyboard port
  - Operating System: Windows 95 and above
  - PC Memory: 16 MB RAM
  - Hard Disk Storage: 10 MB (this is for the provided software)

  The computer we used was a Compaq® PC with 600MHz Pentium III CPU, 512 MB RAM, Windows XP Operating System, 20GB hard disk storage and the required ports. It has reasonably high competence of running the MIMIO software as well as our own application. The software provided by the MIMIO which is used in our tool is discussed in the next section – '*Software component*'.

  The performance of the MIMIO, especially the use of the MIMIO pen is not very satisfactory, which is discussed in chapter 6 – 'Evaluation'. We expect more experiments will be carried out with other similar products to find a better pen-based input device in the future.

- **LIDS** – refers to the Large Image Display Surface unit that consists of: 1) a light metal frame which can be dismantled, with sailcloth surrounded forming a big box (darkroom) which contains some of other devices; 2) a thin screen on the front top side of the box as the drawing surface; 3) a personal computer (PC) set inside or out side the frame which gets digital input from the MIMIO and runs MIMIO software and the application of the tool; 4) a projector set insides the frame which gets digital data form the PC via a connection cable; and 5) a mirror on the back inside the frame which reflects the image of the PC screen projected from the projector onto the displaying screen.

The advantages of this structure over other tools' are the facilitation of the mobility (the frame can be dismantled and shifted), the saving of the space for holding a commonly positioned projector (at least double distanced than the reflected projection), and the elimination of the shadow which will be caused by a common projection (designers will be at the middle between the projector and the screen). However, there are also disadvantages of it such as the need of the relatively complex setup and calibration. The overall usability of the LIDS is quite satisfactory according to our evaluation of the UML sketching tool. At the time of writing this thesis, a new version of the LIDS has been made available to our research. Unfortunately we do not have time to experiment with it and have to leave it as possible future work.

Software Components

In this section, we discuss the software components which carry out the core functionalities required by the tool. Here, we only focus on the basic functions they provided and the interactions between themselves and external components/systems. The detailed implementations of algorithms/mechanisms adopted to achieve the tool functionality are presented in next two sections.

- **UI Manager** – this component deals with all user interface related issues. The MIMIO Mouse software supplied by the MIMIO is used to turn the MIMIO stylus acting as a conventional mouse. When the screen of the computer is projected onto the surface of the E-whiteboard, the user can use the stylus, which is now the MIMIO Mouse to perform most of the actions of a conventional mouse, e.g. quickly press twice on an icon on the desktop to start an application which simulates the 'double click'; press and hold on an icon on the desktop and then moving it into the 'Rubbish Bin' and then release the stylus which simulates the 'drag-and-drop'. However, not all mouse actions are supported by the MIMIO

Mouse, e.g. the ‘right click’. This limits some interaction styles in our interface design (e.g. use of pen-tap on modality buttons rather than in-context pop-up menus).

Visual Basic’s UI library is used to process the data of the pen interactions that is captured and translated by the MIMIO Mouse. A line is drawn on the ‘drawing-canvas’ of the user interface in a preferred color and drawing width/style which can be specified by the user prior to the drawing, so that the simulated ink appears on the screen followed the movement of the pen which the user feels it is drawn by him/her.

The data of the pen interactions is passed to the Repository component for further processing. Here we discuss how pen manipulations are handled in our application.
Each manipulation action is triggered by a special pen interaction with the surface of the E-whiteboard, and the ‘*Time-out*’ technique is used here. For Example, the pen press the surface simulates a ‘*Click’*. If the first ‘*Click’* is within a sketch’s boundary, the second ‘Click’ will trigger the ***Copy*** manipulation– to remove the sketch and redraw a same one at the new position defined by the second pen ‘*Click’*. Two immediate ‘*Click’*s simulating the ‘*Double Click*’ will trigger the ***Delete*** manipulation. The ***Move*** manipulation starts when the pen resides on one point with a sketch’s boundary for a curtain time period, the sketch is redrawn along the way when the pen moves, and it stops as soon as the pen leave the surface of the screen. This process simulates the ‘*Drag-and-Drop*’ mouse action. The ***Replace*** manipulation is start when the pen resides on the point close enough to the start point of a sketch for a curtain time period and ten start to draw another sketch on top of the old one. After the new sketch is drawn, the old one underneath it will be removed. The implementation of the Replace is somehow more complex than others and will be illustrated by an example in next chapter.

- **Repository** – this component takes in the data of the pen interactions and then, organizes it into a hierarchical structure if the input is a sketch, or hands it back to the UI Manager to perform an appropriate action if the input is a pen manipulation. The structure of the Repository and mechanisms of providing these functions are discussed in the 'Mechanisms' section later in this chapter. This component also provides functions of: session control, history navigation and undo/redo facilities. Sketches are passed to the Recognition Engine to be recognized, or the Utility component to be saved or exported. Saved sketches can be reloaded by the Utility component and retrieved by the Repository component to be re-processed.

- **Recognition Engine** – this component carries out the major function of the tool – recognition. It contains three sub-components: the 'Single-stroke Recognition' which recognizes text, the 'Multi-stroke Recognition' which recognizes UML icons, and the 'UML Semantics Constraints' which checks the recognized UML icons against the UML semantics constraints. The details of algorithms and mechanisms employed in this component are discussed in next two sections.

- **Training** – this component provides services required by Rubine's Single-stroke Recognition algorithm used in our tool. The algorithm has been introduced in chapter 2 – 'Background Study'. The data of example gestures passed from the UI Manager component is processed by the algorithm contained in this component, and then saved to the database. The data in the database will be retrieved by the Single-stroke Recognition component when performing text recognition. The implementation details will be further discussed in later of this chapter.

- **Utility** – this component provides the services of: to save sketches passed from the Repository into files which are stored in the secondary storage (e.g. the hard disk of a PC); or to load saved files and then translate back to the Repository for the modification; or to export formalized UML diagrams to the external UML CASE tools. The XML encoding technology (please refer to [XML 03] for information about the XML) is used to achieve these functional requirements. The implementation details are also discussed in later of this chapter.

Our application was written in Visual Basic 6.0. This was because we felt that the use of Visual Basic UI library greatly eased the tasks of user interface build-ups and image processing/rendering, which enabled us to spend more time on implementing the algorithms/mechanisms and the data processing. The performance issue will be addressed in the 'Evaluation' chapter.

Data storage

The files saved by the Utility component are in XML formal.

The database is implemented as a Relational database (please refer to [Riordan 00] for information about the Relational database) which is the most widely adopted database type in the software development community. Microsoft's Access® (Office XP®) [Microsoft 03] was used as the database engine for this small sized database in our tool.

4.3 Recognition Algorithms

As the recognition of the sketches is the core function required by the tool, adopting an efficient and suitable algorithm is therefore very important to the success of the tool development. We need to decide on whether we can use existing algorithms proposed by other researches, or refine/combine one from existing algorithms, or even create a new one ourselves. Obviously the first choice is the most efficient one. The two available algorithms have been discussed in our background study with each has a different approach of performing the gesture recognition. We summarize the advantages and disadvantages of them below.

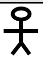
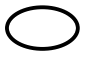
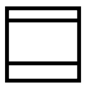
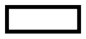
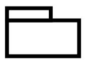
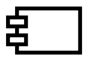
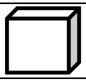
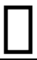
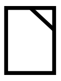
As we have alreafy introduced in the background study, Rubine's **Single-stroke Recognition Algorithm** has been adopted by most similar applications. *Advantages* of this algorithm are: 1) the stroke can be in any shapes 2) has proofs from some implemented applications. However, it has some significant disadvantages: 1) a sketch has to be drawn in one stroke which is not natural for performing UML designs on a whiteboard and is also difficult to handle some complex UML notations, 2) considerable per-gesture and per-user based training is required to gain recognition accuracy which is inconvenient and inefficient for using the tool. The **Multi-stroke**

Recognition Algorithm proposed by Apte *et al* has advantages and disadvantages just opposite to those in Rubine's algorithm.

Due to the complexity of the UML diagrams, we felt that no single approach is a complete solution to recognize sketched UML notations. However, during our research, we found that UML notations could really be categorized into two sub-elements: the **Icon** and the **Text**.

The **Icon** is the graphical part of an UML notation and can be further abstracted as a single or combined geometric shape(s), i.e. *Line, Circle, Ellipse, Rectangle, Triangle and Diamond*. The **Text** is a notation's identifying/specifying text, e.g. *Name, Property, Method, Message, Note, etc*.

We feel that the abstraction of UML notions into geometric shapes is a unique approach in our application. In addition, it can be seen as a generic method which may suit many other notations in different modeling systems, and thus adds a high extensibility to our tool for possible future developments. Table 4.1 summarizes the categorization of notations we implemented in our application.

	Icon	**Shape(s)**	**Text**
Major Notations			
Actor		1 Circle + 1 Horizontal Line + 1 Vertical Line + 2 oblique Lines	Name (1)
Use Case		1 Ellipse	Name (1)
Class		1 Rectangle + 2 inner Horizontal Lines	Name (1) + Property (0 to Many) + Method (0 to Many)
Object		1 Rectangle (Width > Height)	Name (1)
Package		2 Rectangles	Name (1)
Component		3 Rectangles	Name (1)
Node		1 Rectangle + 5 Lines (forms a cube)	Name (1)
Activation		1 Rectangle (Height > Width)	–
Note		1 Rectangle + 1 inner oblique Line at the top right corner	Description

Relationship Notations			
Bi-direction Association	——	1 Line	Name (0 or 1) + Multiplicity (0 to 2)
Aggregation	——◇	1 Line + 1 Diamond	
Dependency	- - - -	1 dotted Line	
Generalization	——▷	1 Line + 1 Triangle	
Message or Single-direction Association	——>	1 Line + 2 short oblique Lines (Arrow)	Message or as above

Table 4.1 Notation Categorization

We further discovered (by our own experiments) that the Icon is relatively easier to be drawn more ‘formally’ than the Text to be written on a whiteboard. For example, the sketches of the Class icon (a compartmentalized rectangle) from two users can be closer than the sketches of the name of the Class from the same two users. This indicates that different recognition algorithms may be chosen to recognize them separately.

Based on above analysis, we decided to adopt a refined **Multi-stroke Recognition Algorithm** for recognize the UML icons, which TWO new parts have been added on top of the Apte’s algorithm.

- *Drawing Rules Based Second-Level Filtering*

  As introduced in chapter 2, the original Multi-stroke Recognition Algorithm contains a filtering technique for recognize simple geometric shapes. However, it is not sufficient for recognize the UML icons which are mostly combinations of geometric shapes. This is especially more difficult when the user sketch them in multi-strokes.

  For example, in Figure 4.4, the (a) and (b) are the Class icon sketched in different orders and stroke numbers, and the (c) is a sketched Note icon. The numeric figures show the ordering and the number of strokes.

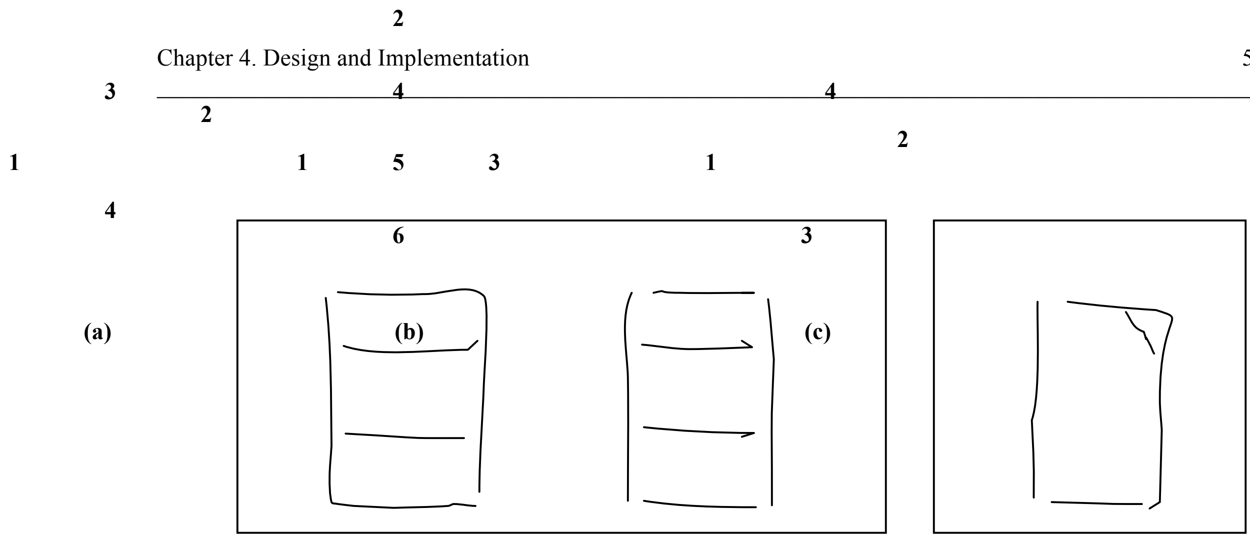


Figure 4.4 Sample sketches: (a) and (b) Class icons, (c) Note icon

If we feed the above three sketches into the filters contained in the original Multi-stroke Recognition Algorithm, the output will the same – a rectangle shape (the algorithm ignores any strokes within the boundary of the sketch. We can see then a **Second-Level Filtering** method need to be applied to distinguish the Class icons and the Note icon. The difference between two types of icon is that the Class icon has two inner horizontal lines and the Note icon has an oblique line at the inner top right corner. However, we do not know which stroke(s) is(are) the identifying stroke(s). Initially we plan to implement a complex algorithm which checks all strokes' shapes and then analyzes their relationships to each other. This may be OK to some relatively simple icons like the two in the above example. However, there is a high cost in terms of the recognition speed if it is applied to some more complex icons e.g. the Component icon and the Node Icon. And yet we had the time limitation for developing our tool. Therefore, a **Drawing Rule** technique is applied which contains 'rules' of how an UML icon has to be drawn by the user. As this is a temporary solution to ease our implementation which is expected to be unsatisfactory to the user, we have done our best to set up these rules which follow the most typical and natural drawing behaviors the users may have.

Firstly, the Drawing Rule defines the order and the number to complete the identifying stroke(s) of an icon. For example, the ***Actor*** icon must be sketched starting from a circle – the head, and then a horizontal line – the arms, and then a vertical line – the body, and then legs which can be draw in one or two line(s). The ***Class*** icon must be sketched starting from the Rectangle which can be drawn in multiple strokes and then two inner lines. The ***Package*** icon must be sketched starting from the larger rectangle which can be drawn in multiple strokes and then the small one which must be drawn in one stroke. And so on.

Secondly, the Drawing Rule defines the position of strokes and the size of the sketch for an icon. The reason of this is explained below.

In Figure 4.5, the user attempts to sketch the Package icons (a) and (b), and the Component icons (c) and (d). The sketches in (a) and (b) follow the rule of drawing order (the head square is drawn at the last) and the number of strokes for the identifying stroke (the head square is drawn in one stroke). However, the sketch in (b) does not look like the Package icon as the head is at a wrong poison with a wrong size. The (a) is recognized as a Package icon, but the (b) is not. This exactly happens to the sketched Component icons in (c) and (d).

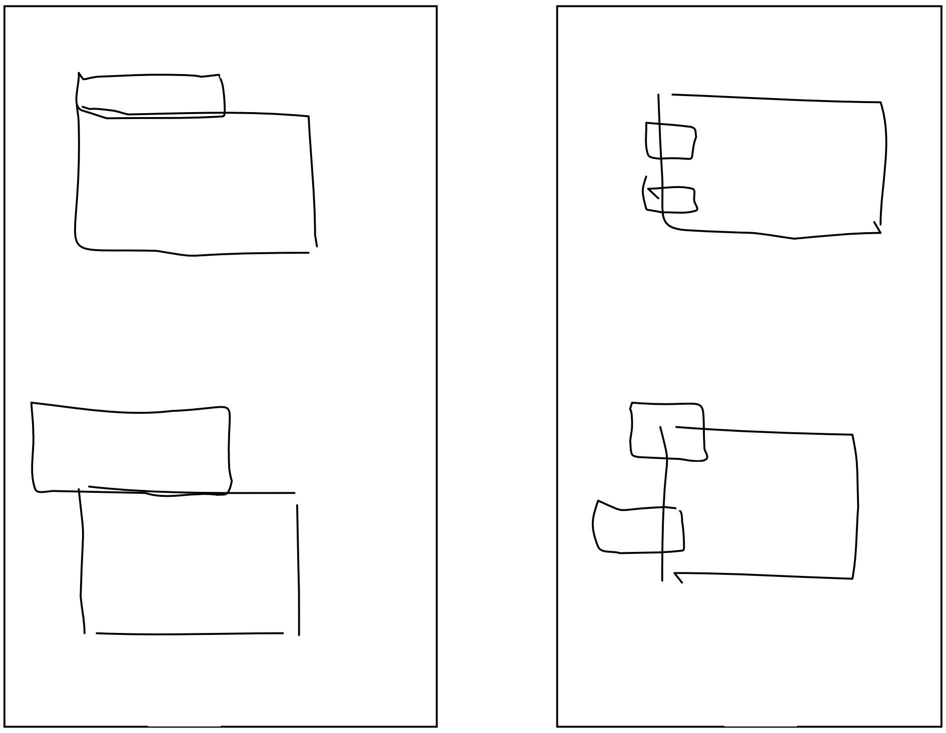


Figure 4.5 Sample sketches: (a) Package icon, (c) Component icons, (b) and (d) not recognized

Thirdly, another rule is set to distinguish UML icons with some accidentally matched non-UML icons. For example, in Figure 4.6, the (a) is intentionally

(a) **(b)**

sketched by the user and should be recognized as the Use case icon, but (b) is drawn with no relation to any UML notations but accidentally matched the Use case shape (Ellipse), and therefore should not be recognized as a Use case icon.

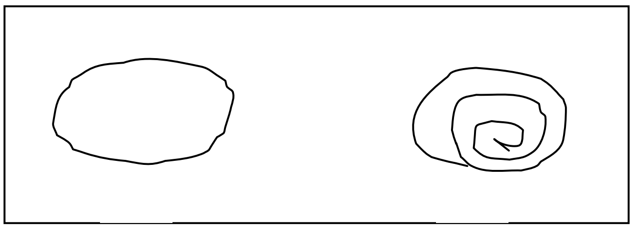


Figure 4.6 Sample sketches: (a) Use case icon (b) not recognized

We used a method which checks whether the total length of the stroke(s) of the sketch matches the perimeter required to form the shape. If the difference is out off the acceptable range then the sketch will not be recognized.
A complete list of the drawing rules is specified in Appendix A.

Based on the above Drawing Rule, the Second-Level Filtering technique is applied for determine the UML icons. This is used in conjunction of the 'first-level' Filtering algorithm of the original Apte's algorithm. The first-level filtering algorithm returns the most possible geometric shape and then the second-level filtering algorithm is used to check if it is likely to be one or a part of the UML notations defined in Table 4.1. This process goes for several ieterations untile a UML notation type or a 'Not Recognized' result is returned. The Second-Level Filtering algorithm is shown below.

Drawing-Rule Based Second-Level Filtering Algorithm

```
1       Is it a Line?
    1.1     Yes. Is it in ONE stroke?
        1.1.1   Yes. → Bi-direction Association relationship.
        1.1.2   No. → Dependency relationship.
    1.2     No. Is it a Semi-Line (nearly to be a Line)?
        1.2.1   Yes. Is it more then TWO strokes in total?
            1.2.1.1   Yes. Check strokes other than the first one:
                1.2.1.1.1   Two short strokes with proper position and angle against the first stroke.
                    1.2.1.1.1.1   Is horizontal and in the Sequence diagram. → Message
                    1.2.1.1.1.2   Else. → Single-direction Association relationship.
```

1.2.1.1.2 Three short strokes with proper position and angle against the first stroke. → **Generalization** relationship.

1.2.1.1.3 Four short strokes with proper position and angle against the first stroke. → **Aggregation** relationship.

1.2.1.1.4 Else. → **Not Recognized**.

1.2.2 No. Go to 2.

2 Is it an Ellipse?

2.1 Yes. → **Use Case**.

2.2 No. Is it a Semi- Ellipse (nearly to be an Ellipse)?

2.2.1 Yes. Is it drawn in ONE stroke or right after the drawing of an Actor?

2.2.1.1 Yes. → **Use Case**.

2.2.1.2 No. → Go to 3.

2.2.2 No. → Go to 3.

3 Is it a Rectangle?

3.1 Yes. Is the width two times or more than two times greater than the height?

3.1.1 Yes. Does the Perimeter matches?

3.1.1.1 Yes. → **Object**.

3.1.1.2 No. → **Not Recognized**.

3.1.2 No. Is the height two times or more than two times greater than the width?

3.1.2.1 Yes. Does the Perimeter matches?

3.1.2.1.1 Yes. → **Activation**.

3.1.2.1.2 No. → **Not Recognized**.

3.1.3 No. Is the number of strokes greater than one?

3.1.3.1 Yes. Check identifying stroke(s):

3.1.3.1.1 The last one is a short oblique line at a proper position. → **Note**.

3.1.3.1.2 The last two are horizontal lines at the proper positions. → **Class**.

3.1.3.1.3 Else. Go to 4.

3.1.3.2 No. → **Not Recognized**.

3.2 No. Go to 4.

4 Is it a Semi-Rectangle (nearly to be a Rectangle)?

4.1 Yes. Is the number of strokes greater than one?

4.1.1 Yes. Check identifying stroke(s):

4.1.1.1 The last one is a Rectangle or Semi- Rectangle with the proper position and size. → **Package**.

4.1.1.2 The last two are Rectangles or Semi- Rectangles with the proper positions and sizes. → **Component**.

4.1.1.3 Else. Go to 5.

4.1.2 No. → **Not Recognized**.

4.2 No. Go to 5.

5 Is the number of strokes equal to either three or four?

5.1 Yes. Is it true that the first stroke a Circle or Semi-Circle, and the second stroke a horizontal Line with proper position and size, and the third stroke a vertical Line with proper position and size, and the rest stroke(s) is(are) with the proper position(s) and size(s)?

5.1.1 Yes. → **Actor**.

5.1.2 No. → **Not Recognized**.

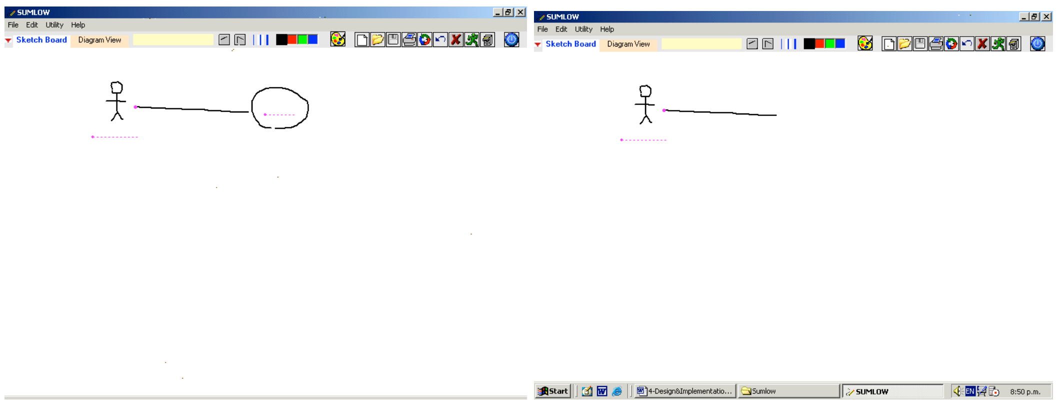
5.2 No. → **Not Recognized**.

- *UML Semantics Constraints Checking*

This is the second part we added on top of the original Apte's algorithm. The need of following the UML semantics definition has been discussed in chapter 3-'Requirements Specification'. After an UML icon is recognized, it is then checked against the UML semantics constraints which have also been discussed in chapter 3. For example, a relationship only exists between two major notations and as such, a relationship icon will not be recognized if it is drawn prior to any major notations have been drawn or is floating (not close enough to two major notations).

However, we modified a few constraints for our own reasons.

1) In conventional UML CASE tools, the relationship icon will disappear if either of associated notations is deleted, but it will not in our tool (Figure 4.). The reason for applying this is that we intend to avoid repeating works to the user, as we have observed that usually the user will sketch another notation to replace the one just been deleted and will rather want the relationship icon being kept in stead of having to draw another one.



(a) **(b)**

Figure 4.7 The relationship (a) between an Actor and a Use Case; and (b) still exists after the Use Case has been deleted

2) We allow different UML diagram types to be sketched in one drawing place, except the Sequence Diagram due to its complexity and the need of significant sketching space (Figure 4.10). The reason for implementing approach is that we felt it provided more flexibility to the users during the conceptual design stage

where they usually want to express their design ideas without too many restrictions.

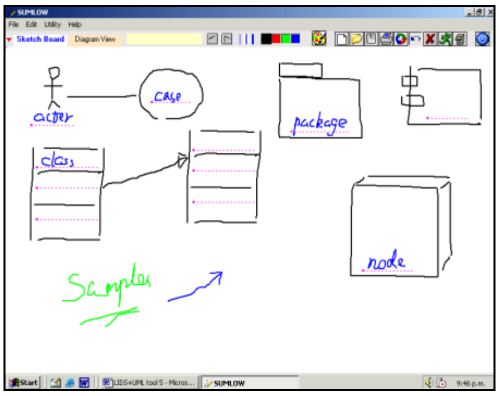


Figure 4.8 An sketching example of the mixture of different UML diagram/notation types

In above, we introduced a refined Multi-stroke Recognition Algorithm for recognize the UML icons. For the **Text** recognition, we chose to adopt Rubine's **Single-stroke Recognition Algorithm** which has been introduced in chapter 2. The reason for us to choose this algorithm is due to its immediate availability. From our user evaluation, we can see it is not ideal. We treat it as a temporarily solution for our prototype. We plan to adopt more sophisticated solution to the text recognition in our future work. It is not discussed further here, but some useful notes will be explained in next section.

4.4 Mechanisms

In this section, we present mechanisms implemented in the tool which carry out its required functions. [illegible]) in a UML Class diagram refined from the conceptual object model presented in previous chapter. This object model illustrates the infrastructure of the mechanisms. The following sections are the discussions of each implemented mechanisms and the reasons and advantages/disadvantages for adopting each of them. In each section, the properties and functions of the main objects in the model which serve that particular mechanism are also discussed in groups.

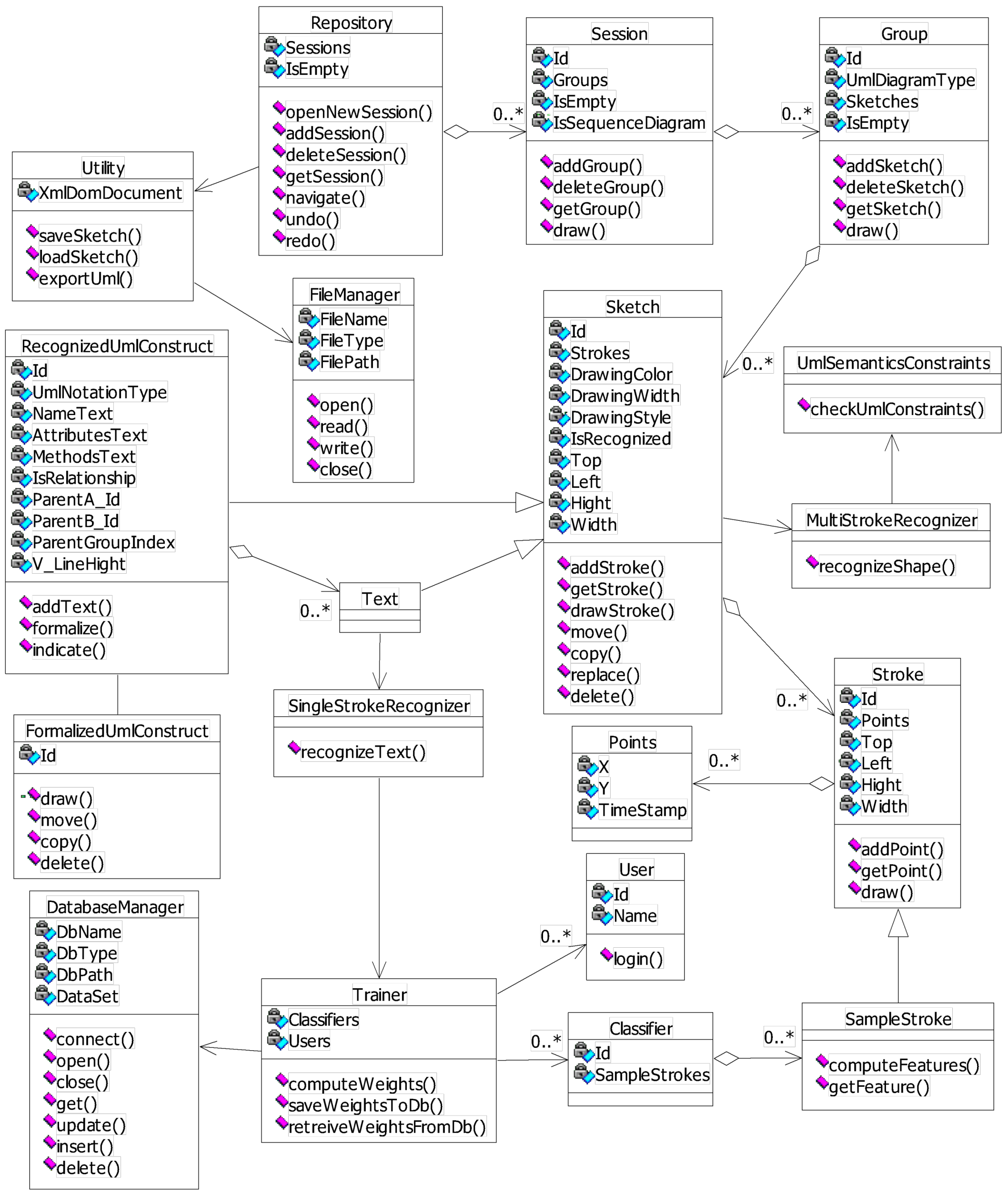


Figure 4.8 Detailed Object Model

Repository Implementation

The Repository is one of the most important mechanisms implemented in our tool. It involves half of the objects in the model (*nine* out of *eighteen*). The functions it services have been discussed in the 'Architecture' section of this chapter. Here we discuss its hierarchical tree structure which starts with: 1) the **Repository** object – the root of the tree which manages the work done within the life of the application (from the open of the application tile the close of the application). It contains subsets of, 2) the **Session** objects which manage the work done within each drawing place. The history of sessions can be navigated. Pen manipulations to one or all sketches in a session can be undone or redone. The subsets it contains are, 3) the **Group** objects which refer to different *UML diagram* types plus the *Secondary* notation type. The next level is, 4) the **Sketch** object which is the major object in this structure. It has special types of, 5) the **RecognizedUmlConstruct** object representing the recognized sketch and, 6) the **Text** object which is also an aggregation type of the **RecognizedUmlConstruct** object. The **RecognizedUmlConstruct** object has an associated object: 7) the **FormalizedUmlConstruct** representing the formalized object of the **RecognizedUmlConstruct** object. The subsets of the **Sketch** object are, 8) the **Stroke** objects which also play important roles in this structure. The Stroke object consists of the leaf object, 9) the **Point** object which is the basic element to store data to be processed.

The *Pen Manipulation* facility relies on the Repository mechanism. However, the operation sequences representing how pen manipulations are performed have been illustrated in previous chapter. There are also other functions supported by this mechanism e.g. history navigation, undo/redo facilities. However, they are not significant but rather general to average applications. We thus do not discuss them here.

Recognition Implementation

The **SingleStrokeRecognizer**, **MultiStrokeRecognizer** and **UmlSemantics Constraints** objects are involved in the implementation of this mechanism. However, most implementation details have been discussed in previous section. We here only discuss the issue of when the recognition should take an action?

In Rubine's algorithm, the recognition starts as soon as a stroke is finished. There is another method called 'Eager' recognition [Rubine 96] which refers the recognition of sketches as soon as they are unambiguous. The approach we adopted is the 'Time-out' technique proposed in Apte's algorithm which has been explained in chapter 2.

However, we have made it to be adjustable to suit different drawing habits from different users.

Training and Database Implementation

This mechanism is required solely by the Single-stroke Recognition Algorithm. The Classifiers contained in the **Trainer** object refers characters we wish to be recognized. The **Classifier** object contains the **SampleStroke** object which is a generalization type of the Stroke object. Features of **SampleStroke** objects supplied per user are computed by the **Trainer** to produce a set of *weights* which are stored in the database and will be retrieved later to be used for the text recognition. The **DatabaseManager** object is responsible for interacting with the database.

We briefly discuss the database structure which is illustrated in Figure 4.8. The **Classifier** table contains classifiers, which are characters used for the text recognition (26 letters and some additional special characters). The **User** table records the data of the different users using this application. The **Weight** table has sets of weights in which each set relates to a particular Classifier and a particular User. When the Single-stroke Recognition Algorithm is invoked, corresponding weights are retrieved to be matched for a newly sketched character for the recognition.

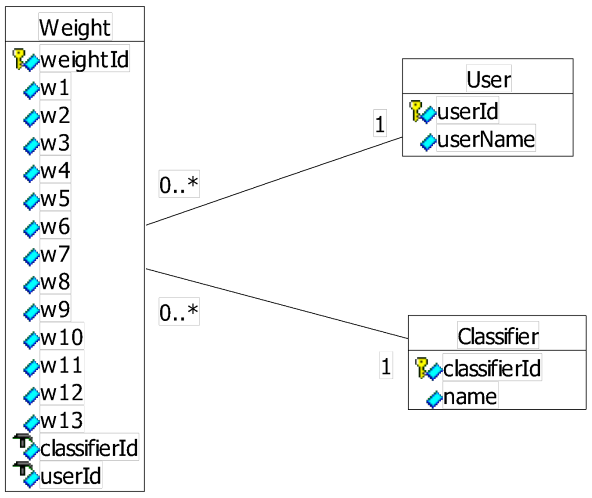


Figure 4.9 The Database Structure

Saving/Loading and Exporting Implementation

The **Utility** object is responsible for performing this mechanism. It transforms the data stored in the Repository structure into the *XML DOM* (Document Object Model) component and save them to an *XML* formatted file. The Loading operation is an opposite process to the above. The **FileManager** object interactions with the file storage system.

The initial implementation plan for the Exporting of UML diagrams is to create a communication channel/method between our tool and a third part UML CASE tool and build the plug-in components to each of the tool to enable the exporting. However, after a few times of review to this approach, we found that could really be implemented using the same XML technology, which the data of the formalized UML diagrams is saved as XML files, and then a build-in component in the third party UML CASE tool will simply read in the data from the file and translates to the UML diagrams of that tool. Though a protocol of the data structure in the XML needs to be agreed between two parties, the new approach is much simpler than our initial implementation plan of this function.

4.5 User Interface

The design and implementation of the user interface followed the consideration points discussed in previous chapter. Figure 4.9 illustrates the structure of the implemented user interface.

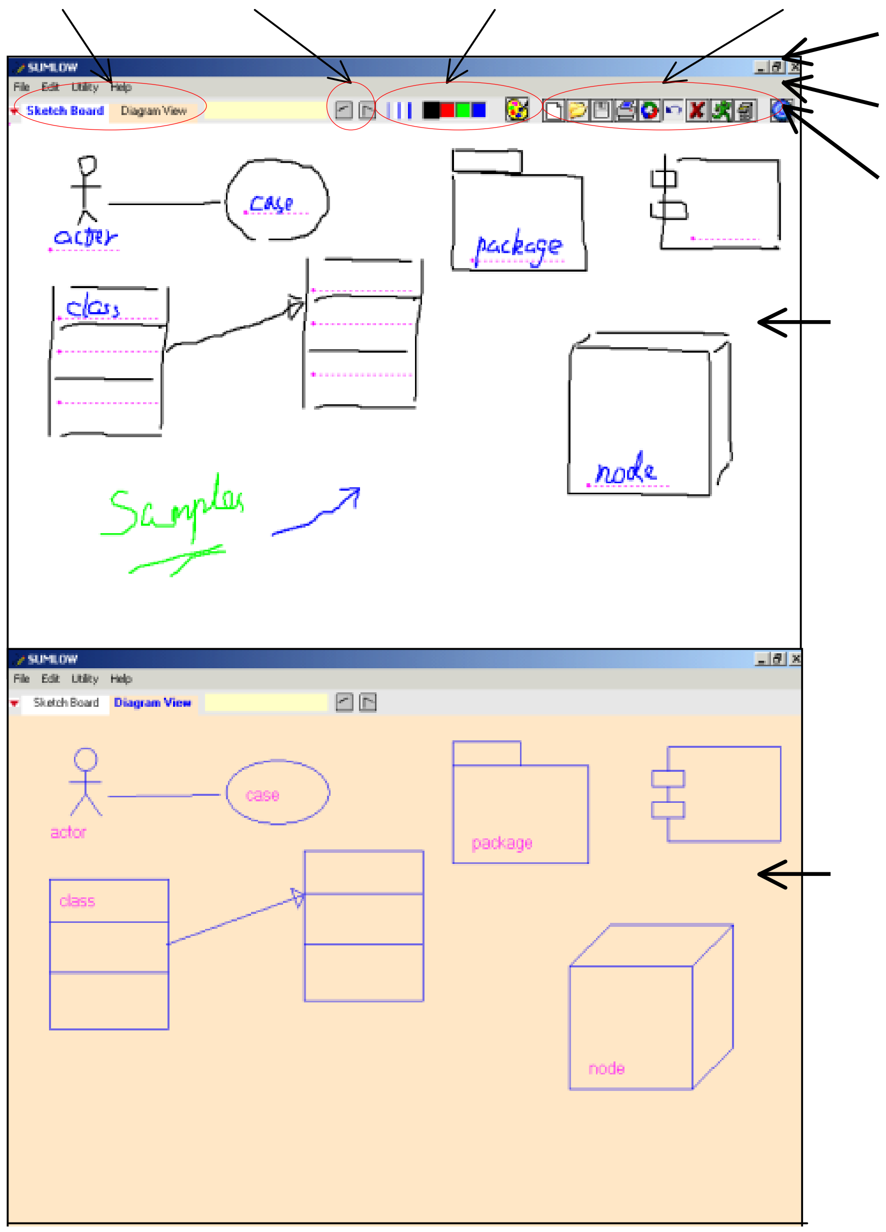


Figure 4.10 The User Interface

Compared with a conventional whiteboard, the screen which appears on the E-whiteboard has three extra thin bars: the windows caption bar, the menu bar and the

The shift of the tool bar

tool bar. The tool bar can be placed in either top or bottom position of the screen (showing in Figure 4.10) to ease the reach for different users. The background color of the ‘Sketch Board’ (a drawing place containing all sketched UML notations and secondary notations) is white to simulate a real whiteboard. The ‘Diagram View’ (a view only contains formalized UML notation) is in a different color to be distinguished from the “Sketch Board’. Two views can be easily switched by clicking two labeled tabs. More examples of the effect of ‘look and feel’ will be shown through a case study in next chapter.

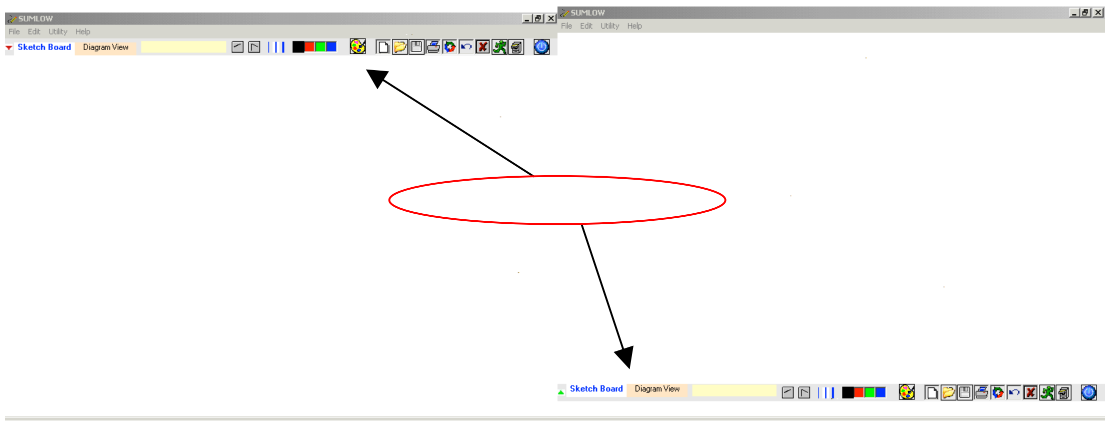


Figure 4.11 User Interface

4.6 Implementation Experiences

Reviewing our implementation, we found that the ‘Saving/Loading and Exporting’ part is the easiest one as we used the new XML technology. It could be very hard to be implemented with other approaches.

We feel the hardest part is to find the appropriate and efficient recognition algorithms. Many efforts have been spent on this issue. We are very happy with the refined Multi-stroke Recognition Algorithm for recognizing UML icons. Our own work added on top of other’s work has been proved. However, the Single-stroke Recognition Algorithm for the text recognition which almost ‘copied’ from published resources is not satisfactory. This experience indicates that, while we are trying to use as many outcomes from others’ work as possible to avoid ‘reinventing the wheel’, to always seek possible improvements to others' work is still an important and right manner to a research.

We are also happy with the Repository management facility we implemented. It provides unique characteristics to our tool and an effective and efficient structure for handling the storage of all data being processed for the tool functionality.

We have learned that to design a 'correct' architecture will greatly ease and shorten the implementation process as a clear guide has been provided which can be followed by the developers to build their desired tools.

We have also learned that the reviews and refines to the initial implementation plans are necessary for adopting better solutions. This has been proved by the example of implementing the *Exporting* facility in our tool development.

4.7 Summary

In this chapter, we discussed details about the design and implementation of the tool development. The discussion covered aspects of the design of the architecture and user interface of the tool, the recognition algorithms and other mechanisms implemented in the tool, the reasons and advantages/disadvantages of the design and implementation choices we made, and the implementation experiences gained. This chapter then not only comprehensively explained how and why the tool was developed which led to the final prototype, but also provided insights of how an E-whiteboard/pen based sketching tool could be better built.

CHAPTER 5 - A Case Study

5.1 Introduction

We carried out a case study to use our tool on a realistic design example after it has been implemented. It was seen as a part of the testing process in the tool development life cycle, and was also used in the evaluation survey.

In this chapter, we present the case study to provide a visual impact to readers on how the tool is used for sketching UML in software design. To achieve this, it is not necessary to have a comprehensive 'full' case study covering all aspects in software design and all UML diagrams involved. Instead, we focus on three most commonly used diagrams – Use case diagram, Class diagram and Sequence diagram to illustrate the differences between our tool and a general UML CASE tool and the features our tool has over a conventional whiteboard. Some implementation details are also discussed further in this chapter.

5.2 The Case

The case is from John Grundy's tutorial of '*Requirements Engineering, Software Architecture and Object-Oriented Design Using UML'* in the *Software Engineering* course at the *University of Auckland* [Grundy 00]. It is a simple on-line video rental system for a video store. We first list its background information and then present the completed Use case diagram, Class diagram and Sequence diagram as a part of high level model of the system. The analysis steps which led to those diagrams are not discussed as they are obviously out of the scope of this thesis. The diagrams were drawn using a common UML CASE tool: *Microsoft Visual Modeler* developed by *Rational Software Corporation* [Rational 03]. Later in this chapter, designers will sketch some of these diagrams using our tool, and each of them can be compared accordingly.

The **Video Store** has an information system to support recording information about videos the store owns, which is searchable by staff and sometimes customers, and information about which customer is renting which videos. This forms a basic ‘library’ system for supporting finding videos and recording video rentals/returns. Customers are assumed to have on-line access to the video library via the WWW (so they can see what videos are available from home before tuning up at the store), and staff are able to record video rentals and returns by customers using a bar code scanner. In addition, staff needs to maintain customer, video and staff information. Managers of the store need to generate various reports, and some data from the system needs to be exchanged with accounts and inventory systems that might already exist.

After analysis of the user requirements, the main Use cases for the video library system are generated to represent the main interactions of Actors (users and other systems) with the system, and are shown in Figure 5.1.

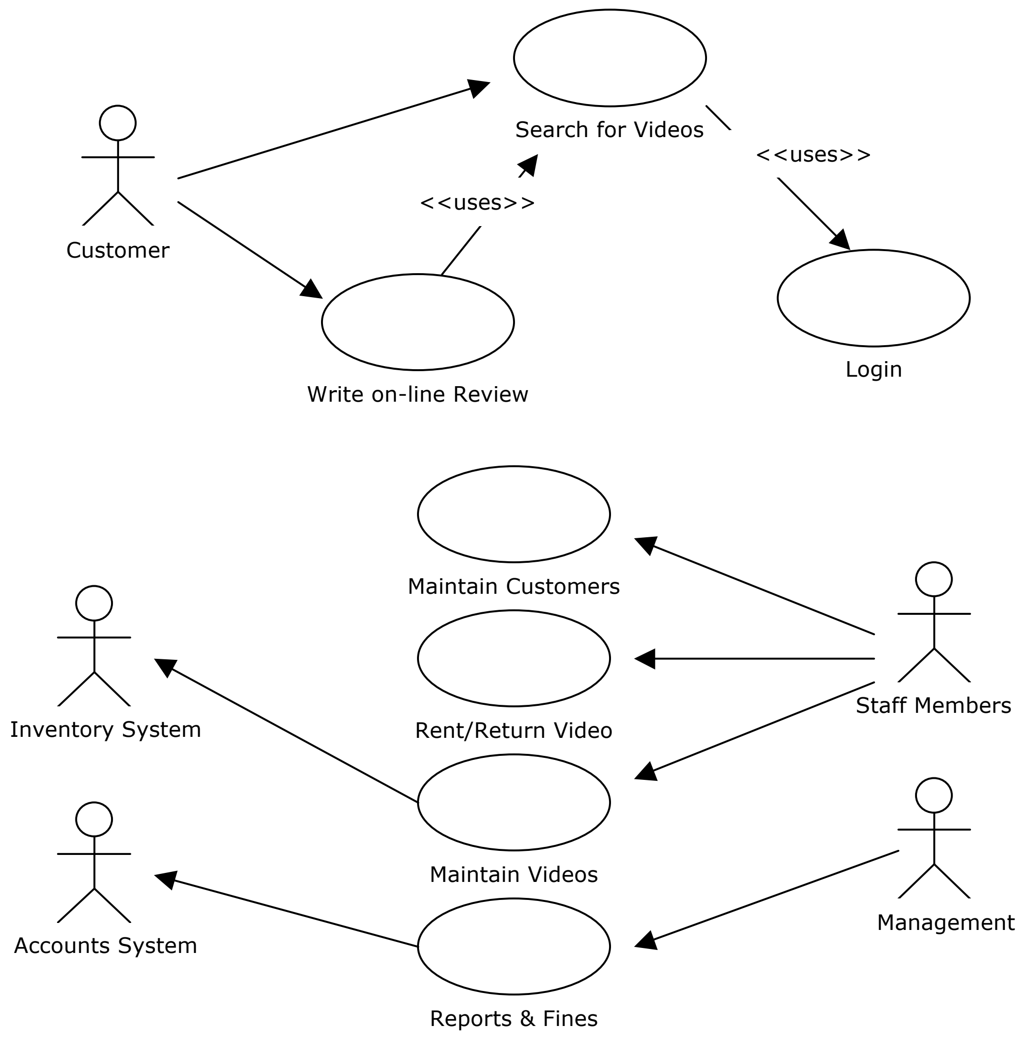


Figure 5.1 The Use case diagram for the video system.

Figure 5.2 is an OOA level Class diagram illustrating the static structure of the system. Each class with its name, properties (attributes) and/or operations (methods), and relationships (generalization, association) are presented in the diagram.

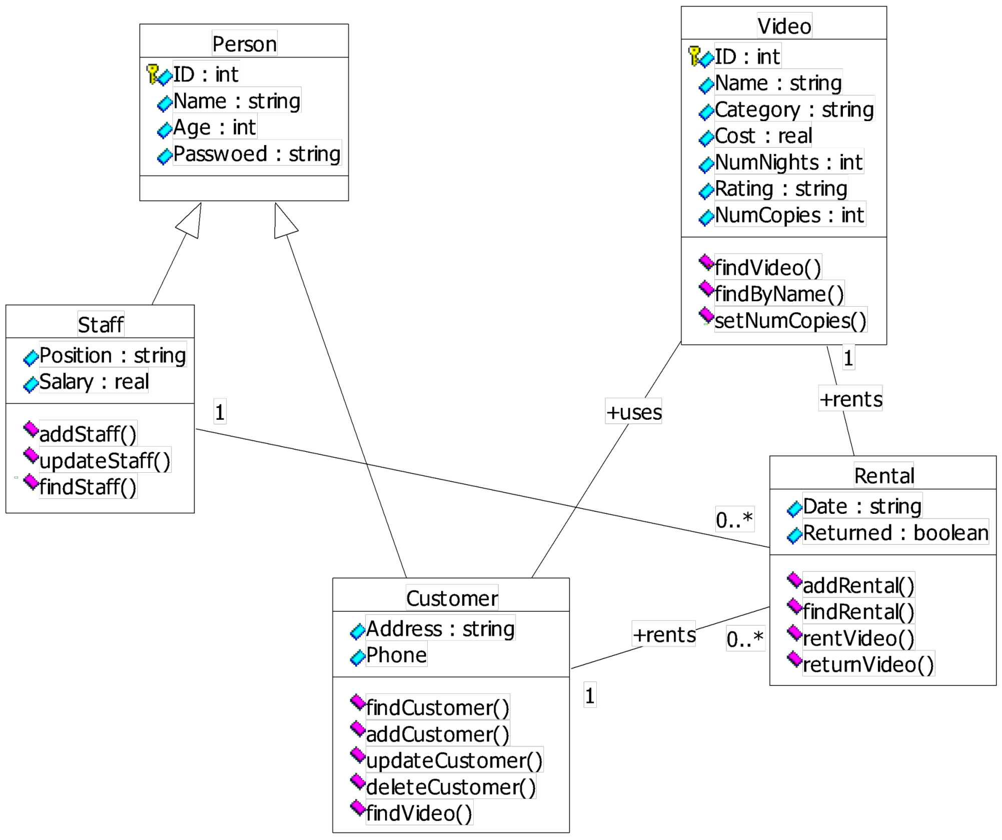


Figure 5.2 The Class diagram for the video system.

In addition, a Sequence diagram in Figure 5.3 shows some of the dynamic behavior of the system. The sequence of operation (function) calls between objects (instances of classes), as well as the arguments to pass and return values are represented in the diagram.

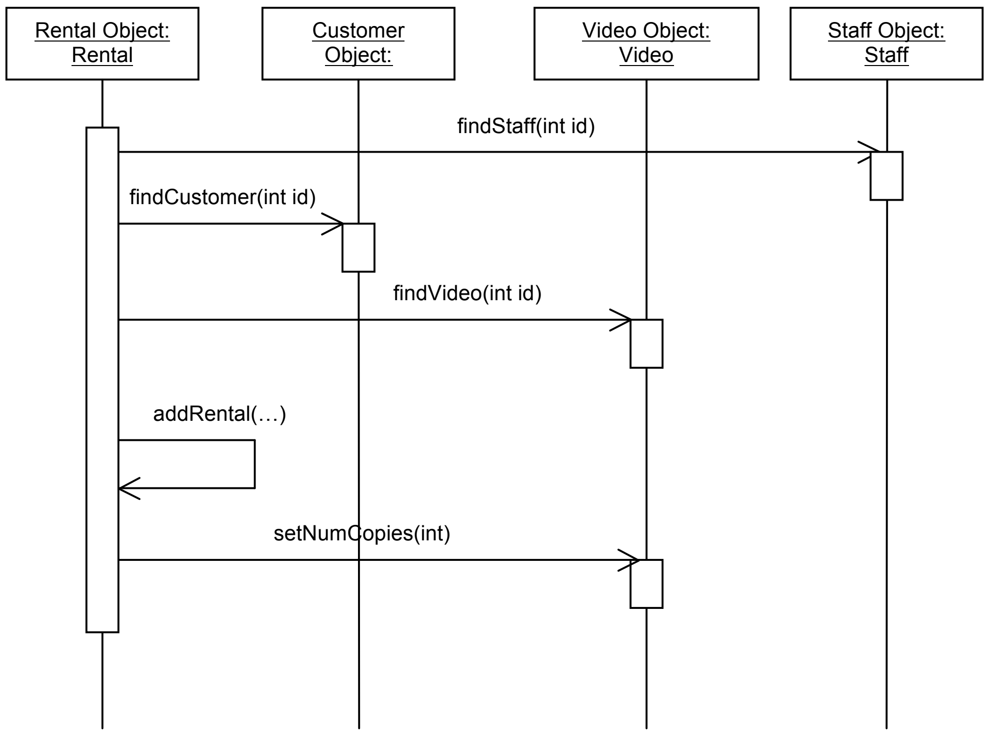


Figure 5.3 The Sequence diagram for the video system.

5.3 The Use of Our Tool

There were two designers – John and Michael who used our tool together to perform early-phase UML designs for the case introduced above.

John begins by using the tool to sketch out the main Use cases diagram of the video system requirements. He draws on the E-whiteboard surface with a stylus and the tool draws connected pixels as John moves the stylus. Figure 5.4, Figure 5.5 and Figure 5.6 show the processes of how the UML icons of an Actor, a Use case and an interaction relationship between them are sketched respectively. We can notice that once an icon is recognized, a text entry area (dotted line) is added to indicate the success of the recognition and also for entering the UML construct's name.

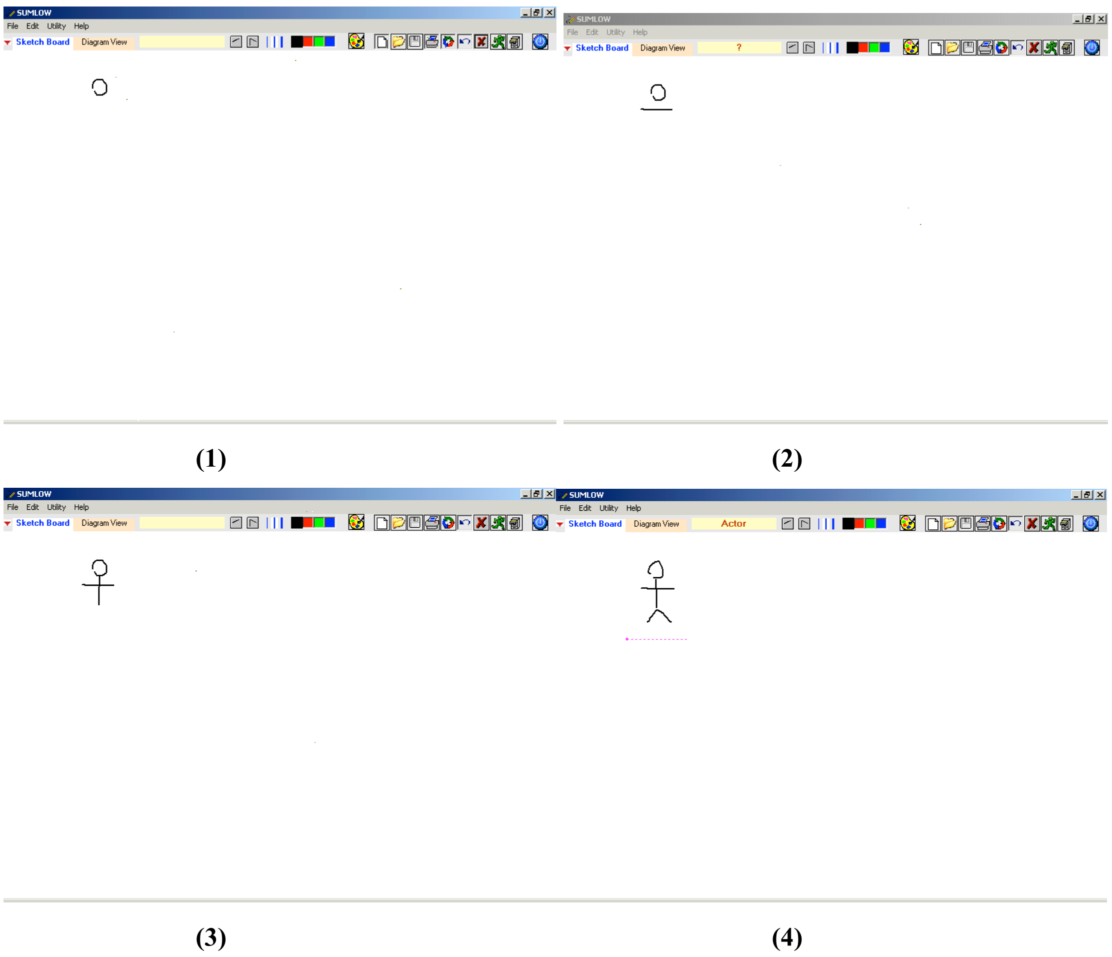


Figure 5.4 The sketching process of an Actor icon – started with (1) and ended with (4)

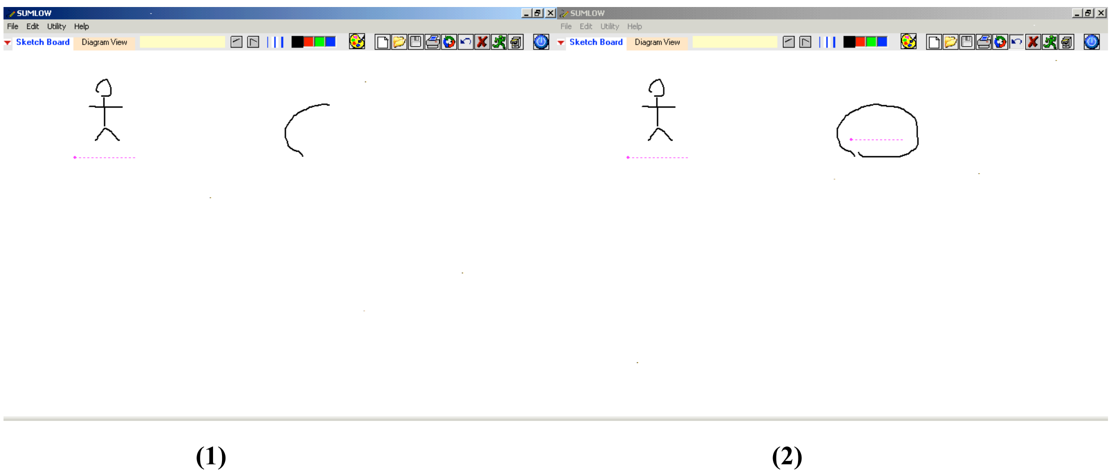
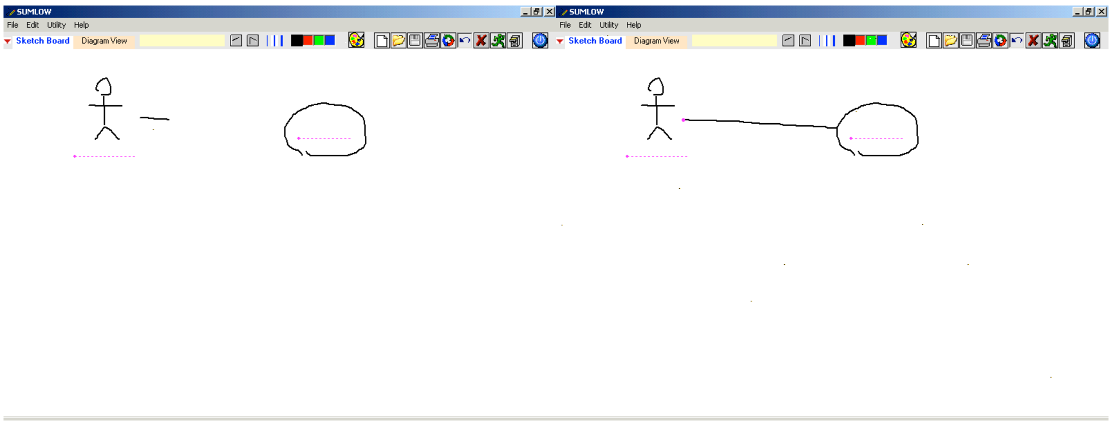


Figure 5.5 The sketching process of an Use case icon – started with (1) and ended with (2)



(1) (2)

Figure 5.6 The sketching process of an relationship icon – started with (1) and ended with (2)

Figure 5.7 shows the resulting Use case diagram in our tool. Both John and Michael have added some annotations, e.g. box around Customer actor, cross through unused use case oval and custom arrow to line, during their discussions of the system requirements.

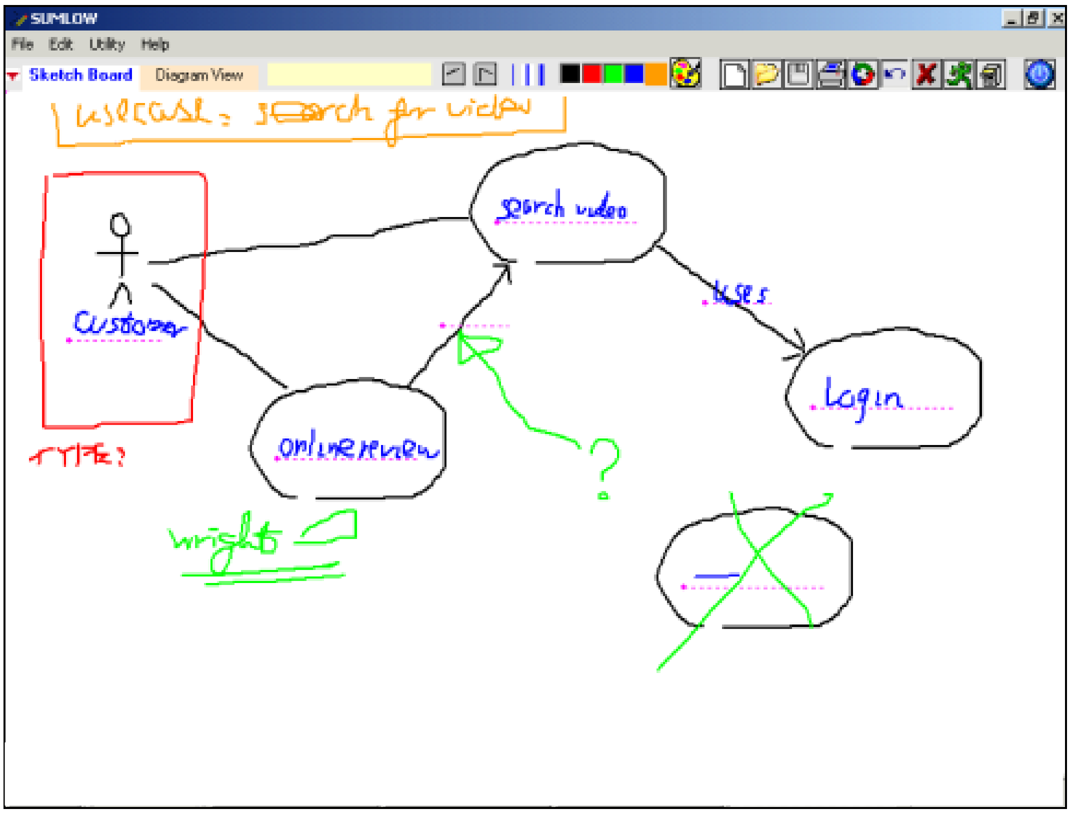
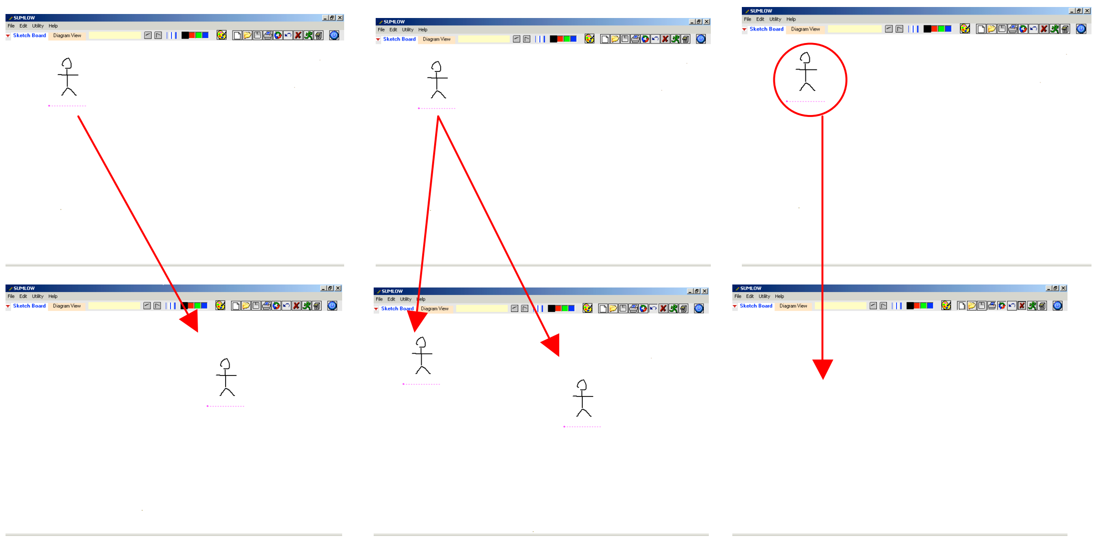


Figure 5.7 The sketched Use case diagram

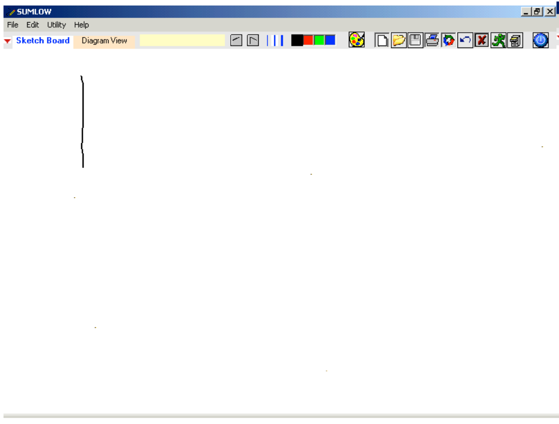
The pen-action based technique (discussed in previous chapter) is used to perform the manipulations of moving, coping, replacing and deleting of the sketches. In Figure 5.8, (a), (b) and (c) show the manipulations of moving, copying and deleting of sketches respectively. The replacing of sketches is illustrated later in Figure 5.10.

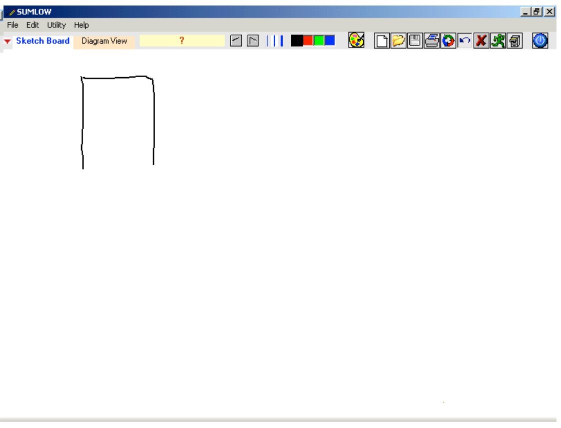


(a) **(b)** **(c)**

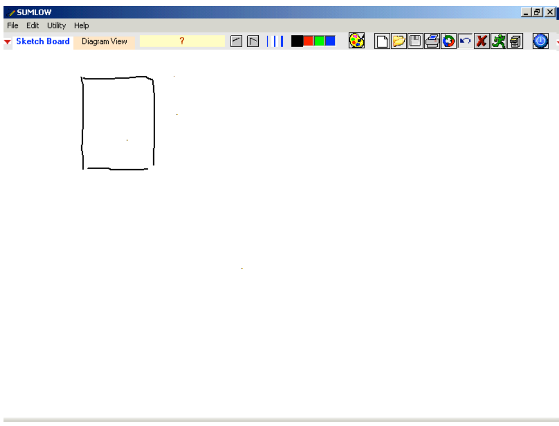
Figure 5.8 Examples of the manipulations (a) moving, (b) copying, (c) deleting

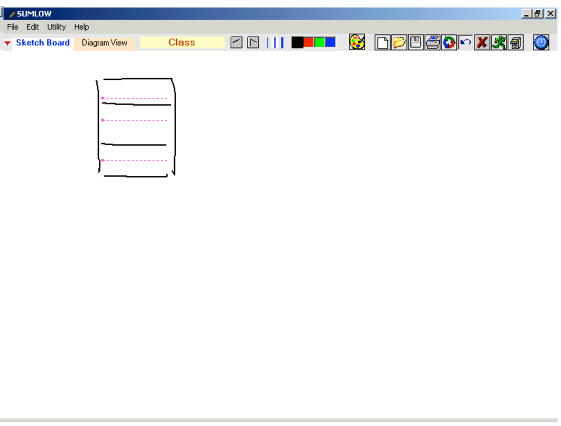
After John has sketched out the Use case diagram, Michael takes over to sketch out some initial Classes (object types) and their relationships. Class shapes are quite complex, being rectangles (that a user may sketch as multiple line strokes) and two horizontal internal lines separating class name (top part), list of class attributes (middle part) and list of class operations (bottom part). Figure 5.9 illustrates the process of sketching a Class icon. The multi-stroke recognition algorithm is used to recognize these and add three text entry areas to the sketched shape, one for each kind of text item the user can draw.





(1) (2)





(3) (4)

Figure 5.9 The sketching process of a Class icon – started with (1) and ended with (4)

Figure 5.10 shows the 'replace' manipulation in action. While Michael writes the name, attributes and operations on those text entry areas of the sketched class icon, the insertion point moves to accommodate additional entries. In this example, it can be seen that Michael has drawn his class too small for the additional textual data, as shown in the top view in Figure 5.10. Rather than supporting a conventional resize operation, a *replace* paradigm is used, whereby the bounds of a construct are redrawn by the user to indicate the size of the replacement, and sub-elements of the sketch are automatically transferred across to the new shape, as shown in the bottom view in Figure 5.10.

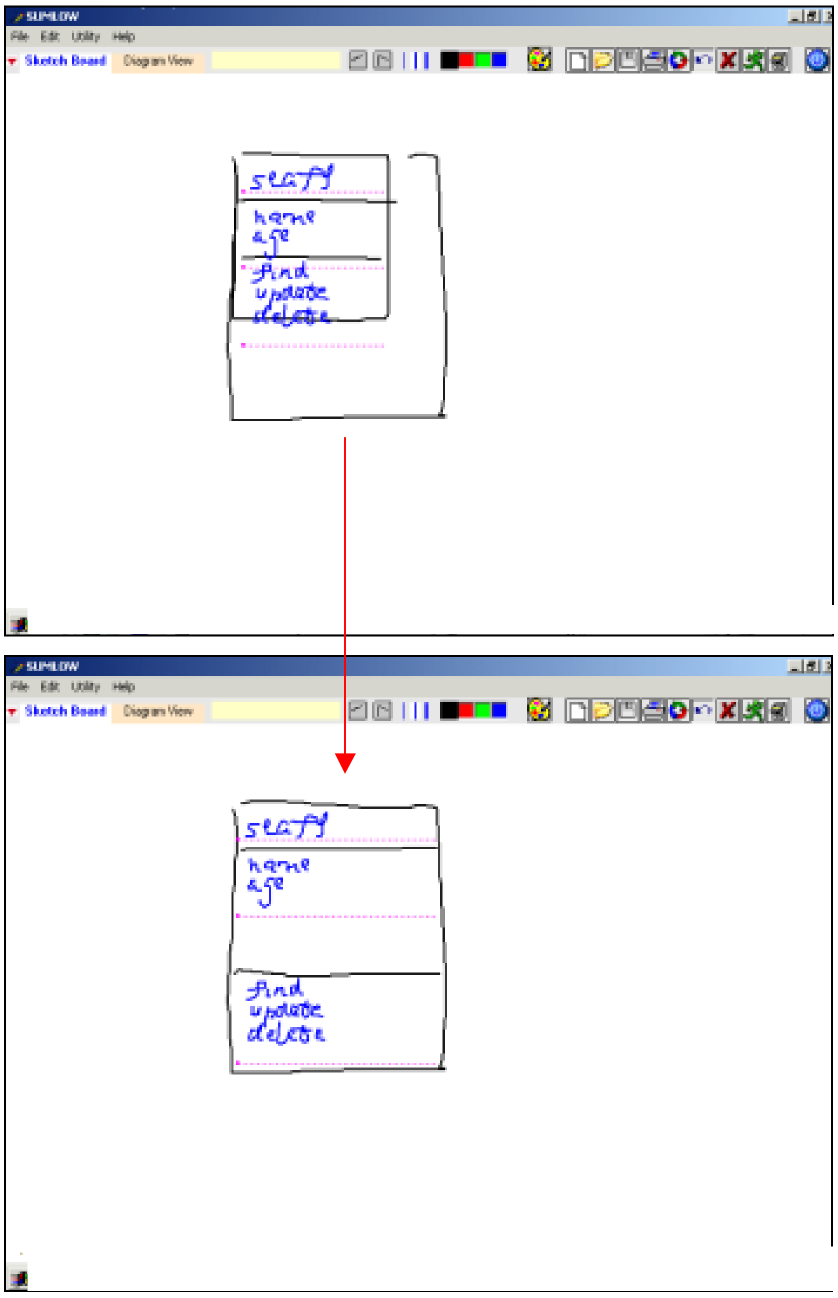


Figure 5.10 The 'replace' technique used to 'enlarge' a class icon

A more complete UML class diagram sketch is shown in Figure 5.11, with several classes, associations (lines between two classes) and generalizations (lines with a triangle arrow). In this example, Michael has named the classes and added attributes and operations to three of them so far. Michael has added an extra use case sketch at the top left (boxed off using secondary notation). From here, we can see while the mixing of different type of diagrams are forbidden in conventional CASE tools like the Microsoft's Visual Modeler shown previously, it *is* implemented in our tool to provide high flexibility for the early stage software design. During the design, Michael and John have also added textual annotation, arrows, and shape highlights which are not recognized as UML constructs and hence regarded as secondary notation.

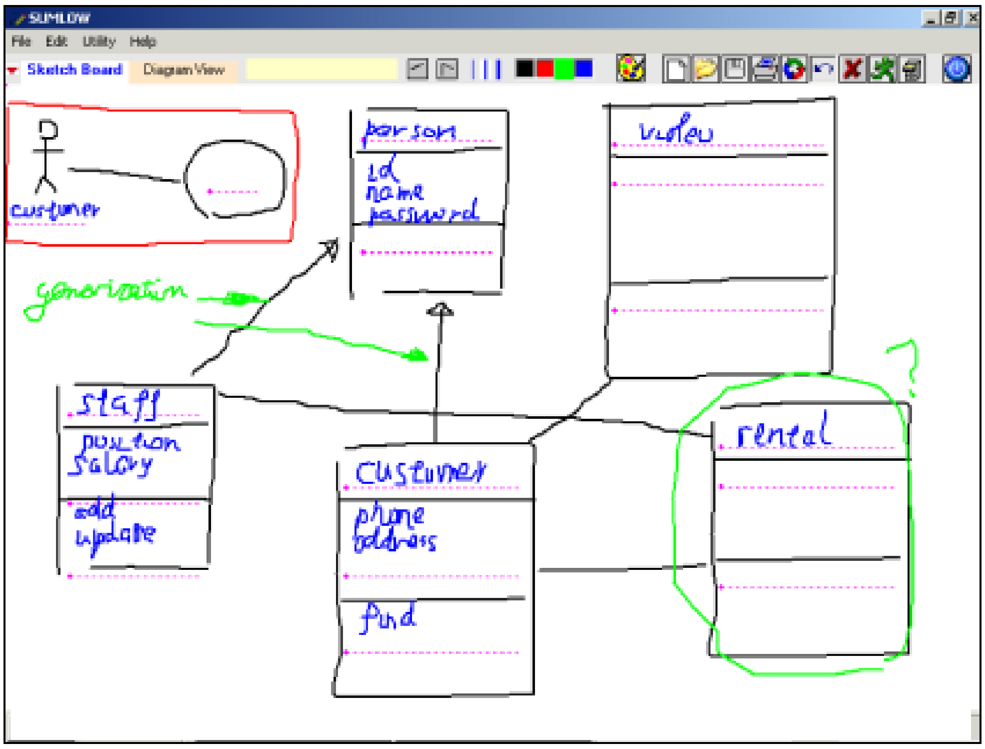


Figure 5.11 Class diagram

After sketching the Class diagram, John and Michael focus on one of the complex message flows in the proposed video system design. They sketch a UML sequence diagram in the tool to capture and discuss this dynamic system behavior.

As sequence diagrams are quite complex and require considerable space, other diagram types are not able to be mixed with a sequence diagram sketch. Initiation of a sequence diagram sketch is done by drawing a horizontal line across the top of the sketch board (Figure 5.12 (a)). At that point, any other existing sketches on the whiteboard are saved or discarded by user choice (Figure 5.12 (b)), and the horizontal line converted to a solid blue line (Figure 5.12 (c)). No matter of the initial sketched position, actors or objects drawn in the sketch board will be automatically relocated at the top of the sequence diagram and a timeline (dotted blue line) added associated with that component (Figure 5.12 (d)). Calls and timing elements are sketched on

(a)

(c)

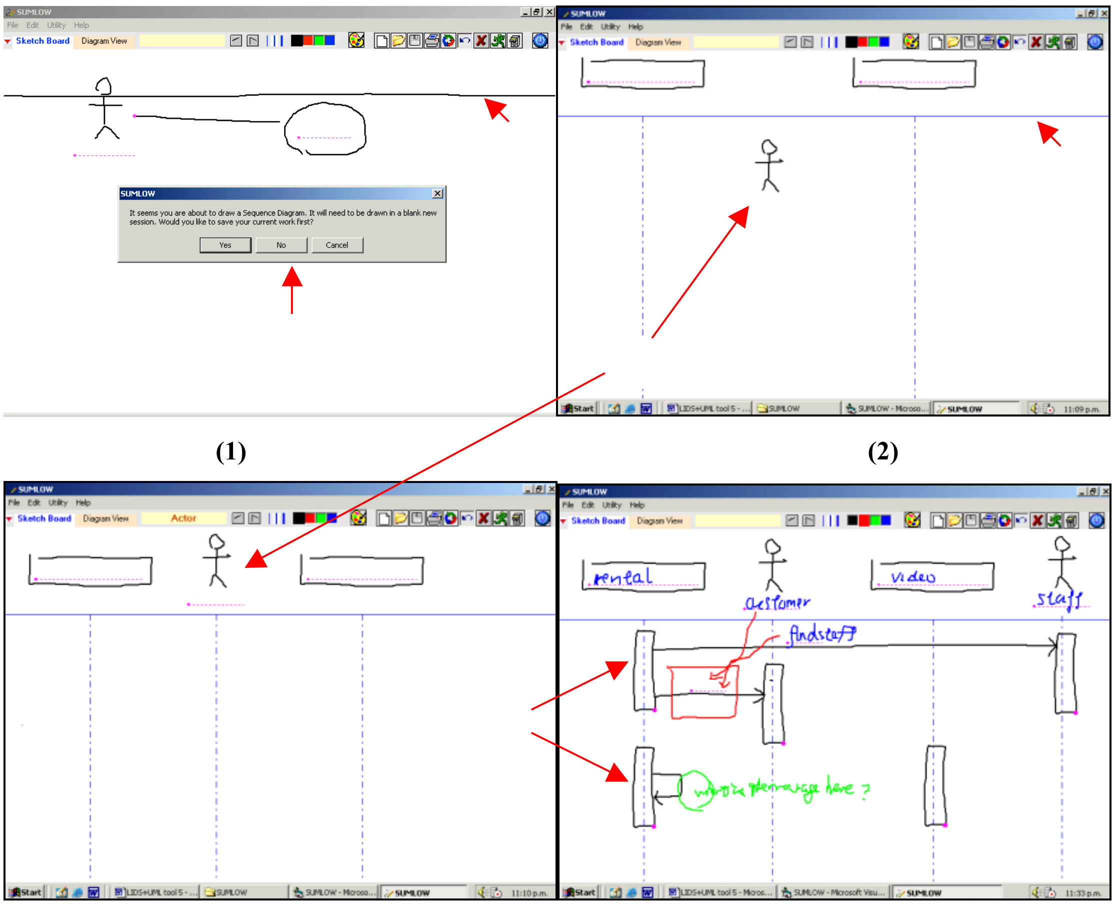
these timelines (Figure 5.12 (e)). Copying, moving or deleting an actor or object will also reposition the timelines, calls, and timing elements as appropriate.

(b) Prompt for saving of pervious work

(d)

Figure 5.12 shows this sketching, with objects (rectangles plus names), vertical lines from objects, operation timing (rectangles on vertical lines), and operation invocation (arrowed lines between operation timing rectangles). In this example John and Michael have also used Actor shapes instead of object rectangles for two objects, customer and staff. This violates the standard UML diagramming convention, but is here useful for John and Michael in discussing their design ideas.

(e)



SUMLOW

It seems you are about to draw a Sequence Diagram. It will need to be drawn in a blank new session. Would you like to save your current work first?

Figure 5.12 The process of sketching Sequence diagram – started with (1) and ended with (4)

As John and Michael perform their design sketching on the *Sketch Board,* sketches are formalized in a background process and rendered into formalized UML diagrams in the *Design View*. The results for some of these sketches are shown in Figure 5.13. Note that all recognized UML construct sketches have been redrawn as computer generated shapes representing standardized UML diagrams of conventional UML

Formalized Class diagram
Formalized Sequence diagram

CASE tools similar to those diagrams shown in the first part of this chapter. All secondary notations are discarded.

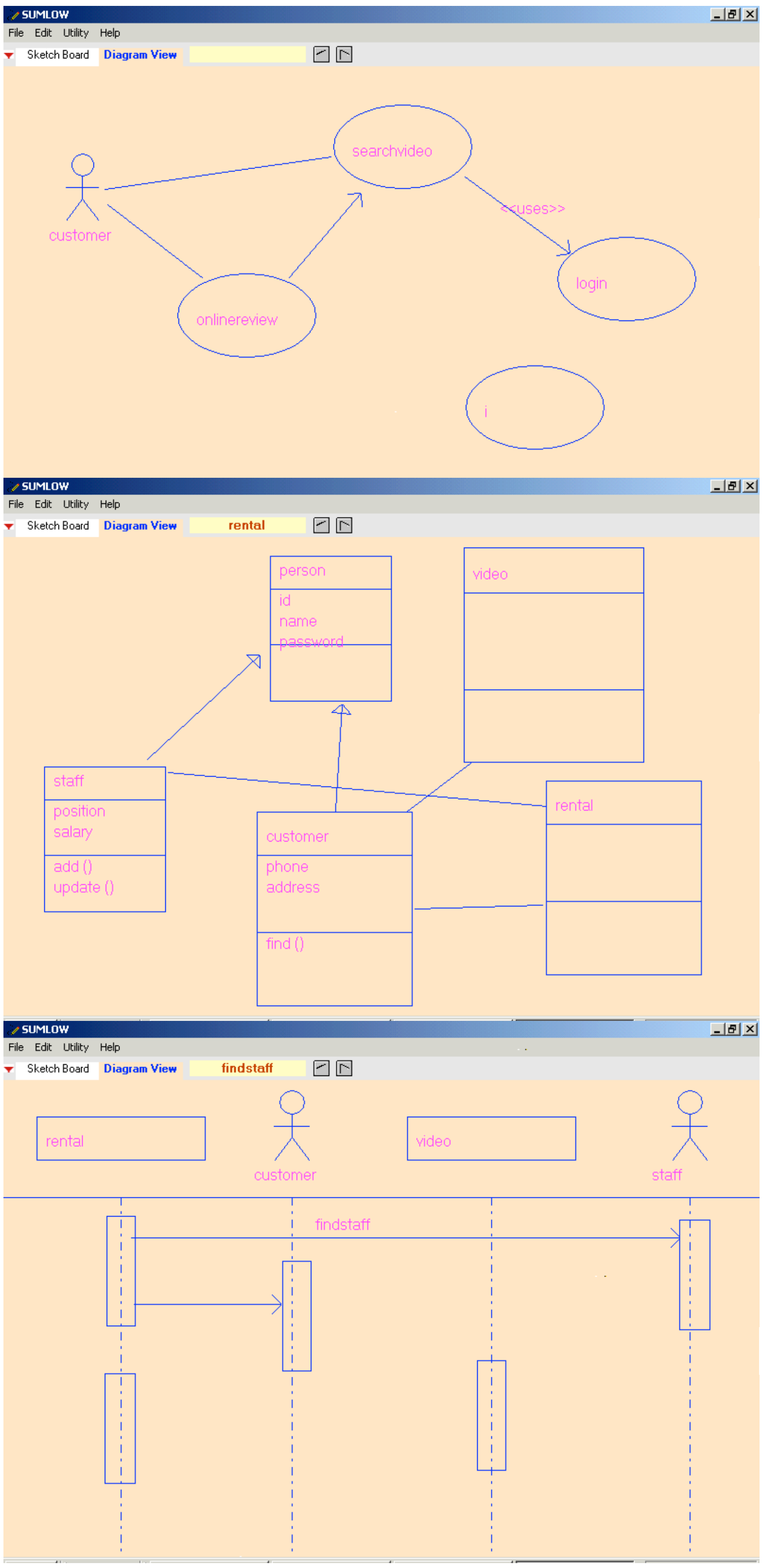


Figure 5.13 "Formalized' UML diagrams from previously illustrated sketches

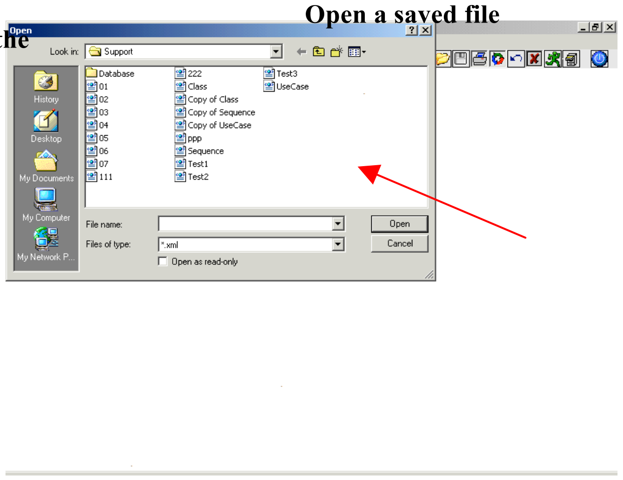
A sketch can only be added (drawn) in the **Sketch Board**. All sketches including UML constructs and secondary notations are remained in the Sketch Board as long as the user desired. This then provides users a least-interruptive environment for sketching and a rich notation support for exploring of high-level (conceptual level) design ideas. Recognized UML constructs are formalized (drawn by the computer) at a corresponding position in the **Diagram View**. Some information is discarded from the sketches e.g. informal secondary notation like highlights that have no UML notation equivalent. Thus, the Diagram View provides a ‘clean’, ‘fine’, ‘formal’ and direct feedback to the user. The two views can be easily switched in between and are completely integrated. UML constructs in each view can be moved, copied and deleted by pen actions and the manipulations will be reflected in another view.

Finally, sketches on the Sketch Board can be saved to a file which can be reloaded (Figure 5.14 shows the reloading process) for the later work. The formalized UML diagrams on the Diagram View can be exported to a 3rd party CASE tool. Both of them are achieved using an XML-based design model encoding XMI. Information of sketches is encoded with the Repository structure discussed in the previous chapter. For the first function, our tool reads in the file and translates data back to the repository. To the second function, a 3rd party CASE tool can load our encoded file and convert it to UML diagrams following a protocol agreed by both parties. Figure 5.15 is an example of information encoded in a XML file for a sequence diagram.

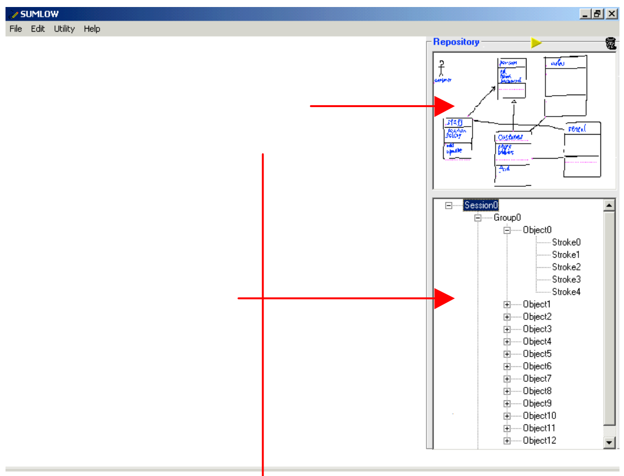
Loaded previous work, click by the pen to add to the current session for editing

Repository viewer showing the tree structure of the data

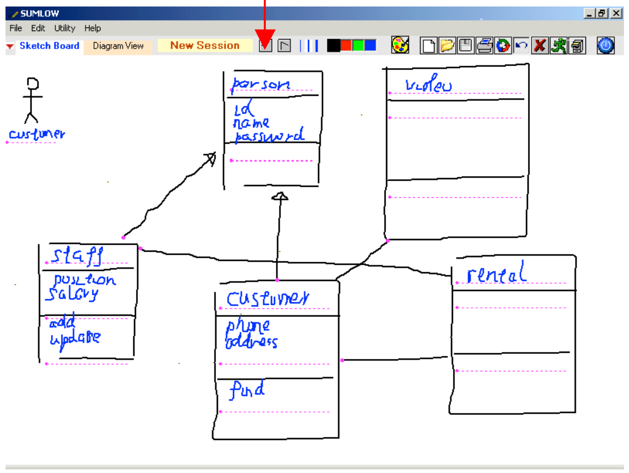
Open a saved file



(1)

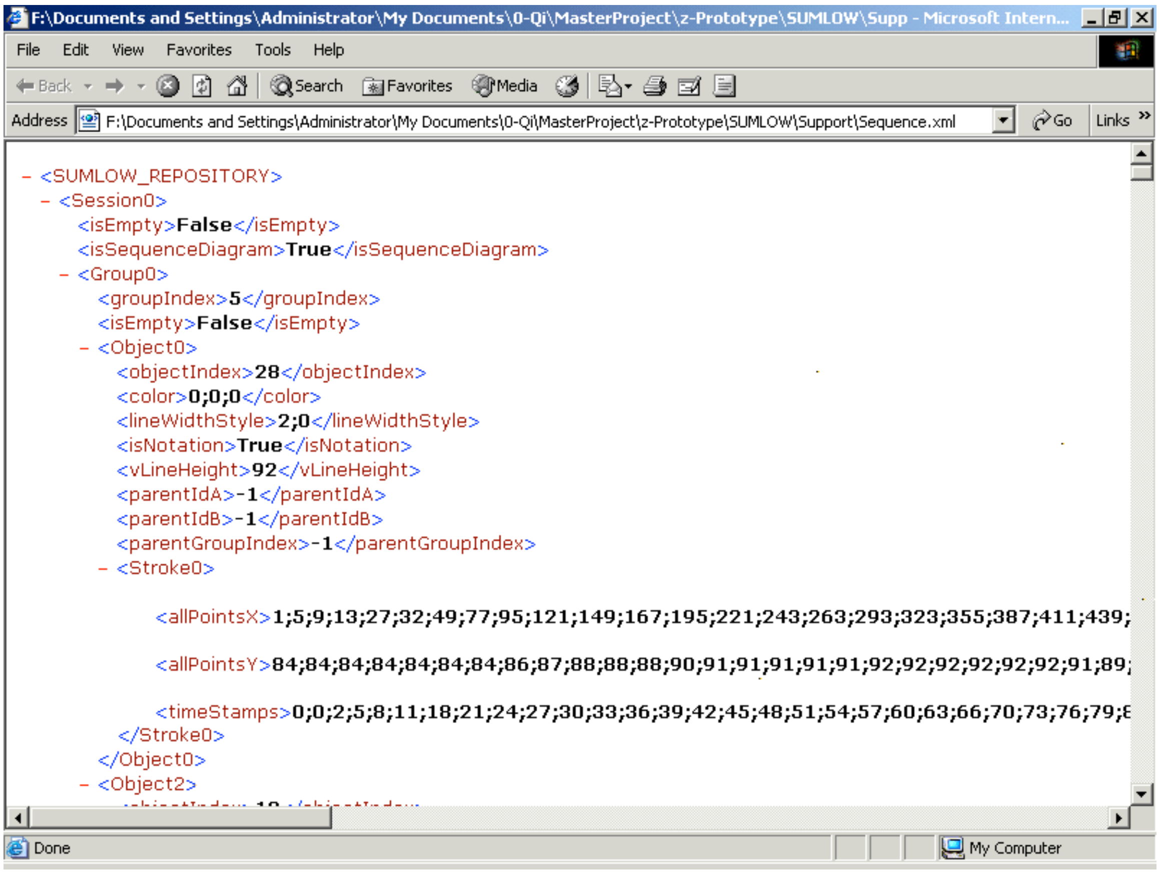


(2)



(3)

Figure 5.14 The process of reloading a saved Class diagram sketch – started with (1), ended with (3)


F:\Documents and Settings\Administrator\My Documents\0-Qi\MasterProject\z-Prototype\SUMLOW\Supp - Microsoft Intern...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address F:\Documents and Settings\Administrator\My Documents\0-Qi\MasterProject\z-Prototype\SUMLOW\Support\Sequence.xml
Go Links
Done
My Computer

```
- <SUMLOW_REPOSITORY>
  - <Session0>
      <isEmpty>False</isEmpty>
      <isSequenceDiagram>True</isSequenceDiagram>
    - <Group0>
        <groupIndex>5</groupIndex>
        <isEmpty>False</isEmpty>
      - <Object0>
          <objectIndex>28</objectIndex>
          <color>0;0;0</color>
          <lineWidthStyle>2;0</lineWidthStyle>
          <isNotation>True</isNotation>
          <vLineHeight>92</vLineHeight>
          <parentIdA>-1</parentIdA>
          <parentIdB>-1</parentIdB>
          <parentGroupIndex>-1</parentGroupIndex>
        - <Stroke0>

              <allPointsX>1;5;9;13;27;32;49;77;95;121;149;167;195;221;243;263;293;323;355;387;411;439;

              <allPointsY>84;84;84;84;84;84;84;86;87;88;88;88;90;91;91;91;91;91;92;92;92;92;92;92;91;89;

              <timeStamps>0;0;2;5;8;11;18;21;24;27;30;33;36;39;42;45;48;51;54;57;60;63;66;70;73;76;79;8
          </Stroke0>
        </Object0>
      - <Object2>
```

Figure 5.15 A screen shot of an opened XML file encoded for a sequence diagram

5.4 Summary

In this chapter, we illustrated the use of our tool through a case study. Rich pictures are provided to present the processes of sketching, manipulation, formalization and saving/loading/exporting of UML designs. We can see that there are significant differences between our tool and conventional CASE tools as well as conventional whiteboards. These different will be further discussed in next chapter.

CHAPTER 6 - Evaluation

6.1 Introduction

This chapter presents an evaluation of the prototype E-whiteboard and pen-input based UML sketching tool we have developed in our research. We focus on evaluating its features and functionality, and aspects of its usability.

First, we apply the *Cognitive Dimensions* (CD) framework [Green et al 96] to evaluate the support of our tool for exploratory UML design compared with conventional UML CASE tools. Then we present a *Survey* of experienced whiteboard users and UML designers we carried out to gain subjective feedback on the tool's suitability for UML based software design. Finally, we make general comments on the strengths and weaknesses of the tool from our experiences with it and its performance characteristics compared to conventional UML CASE tools and conventional whiteboards.

.

6.2 Cognitive Dimensions Evaluation

The CD framework was developed to provide a broad-brush evaluation technique for notation and interactive devices. It proposes a small set of terms that describe different, often competing, aspects of a notation or environment. The terms are meant as discussion tools, highlighting areas of concern that are commonly identified by designers when creating their notations and systems. The framework does not seek to lay down guidelines for the design of cognitive artifacts, but rather attempts to provide a clear basis for discussion and evaluation by pointing out the characteristics of cognitive artifacts that should be considered. We introduce each dimension of the framework and apply them to evaluate our tool below:

- *Viscosity: resistance to change.*

Viscosity is a measure of how difficult it is to perform a local change i.e. how much work the user has to do to implement a small logic change in a model or structure.

The overall viscosity between our tool and other UML CASE tools are at similar level. Changes are mainly happened when manipulating UML constructs. The techniques used in our tool are designed as close to most keyboard/mouse-based tools as possible to provide efficiency and familiarity to users, e.g. the 'click-drag-drop' method to move a UML element. However, some manipulations of sketches in our tool require less effort to change e.g. 1) 'double click' to delete in our tool vs. 'click' and then 'click' a button on the screen or the 'Delete' key on a keyboard, or 'right click' the mouse and then select 'Delete' from the pop-up menu in other conventional UML CASE tools; 2) 'click' on an UML element and then 'click' on another place to perform the 'copy and paste' task in our tool vs. 'click' on a UML element and then press 'Ctrl + C' and then press 'Ctrl + V' on a keyboard, or 'click' on a UML element and then 'click' a 'Copy' button and then 'click' a 'Paste' button on the screen, or 'right click' on a UML element and then select 'Copy' from the pop-up menu and then 'right click' the UML element again and then select 'Paste' from the pop-up menu in other conventional UML CASE tools. While others require more effort e.g. redraw over top to resize in our tool vs. 'click' on a corner and 'drag' in other conventional UML CASE tools.

- *Visibility: ability to view components easily.*

Visibility dimension denotes whether required information is easily available i.e. whether it is or can be readily made visible. It is a measure of how difficult it is to discover and display a piece of information. An important part of visibility is Juxtaposability, which means whether you can display any two separate parts of a model or structure side by side for comparison purpose.

Multiple views are supported in our tool, both sketched and formalized. Users can also mix notations within a view and sketch incomplete UML designs, providing greater modeling flexibility during early design work.

- *Premature Commitment: constraints on the order of doing things.*

  Premature Commitment refers to being forced to make a decision or complete a task before all the information required doing so is available.

  Sketching of incomplete UML diagrams/constructs, and mixed UML diagrams types are supported in our tool to provide higher flexibility during early software design.

- *Hidden dependencies: important links between entities are not visible.*

  A hidden dependency is a situation when a component is dependent upon another component but that dependency is not fully visible.

  The shape recognition algorithms employed by our tool connect hand-sketched design elements to "assumed" formal UML elements within diagrams with limited user feedback.

- *Role-Expressiveness: the purpose of an entity is readily inferred.*

  Role-expressiveness refers to how easy it is to tell what the role of a particular object in a model is.

  Recognized UML constructs are indicated by dotted lines for entering its name/properties)/methods to support role-expressiveness.

- *Error-proneness: the notation invites mistakes and the system gives little protection.*

  Error-proneness is a measure of how likely it is for a user using the language/tool to make a syntactic mistake or slip. That is, not how likely it is they make an error in reasoning but how likely it is they know what the correct thing is to do but simply make a mistake in carrying out the task.

At current stage, for the purpose of simplifying our implementation of the tool, we require users following some ‘rules’ (time between two strokes, order of drawing, the total number of strokes, etc.) to sketch UML elements. For example, to sketch a ‘Package’ icon, the bigger square at the lower position must be drawn first with any numbers of strokes and then the smaller square at the upper left position must be drawn within a predefined time pause (say 0.5 second) with only one stroke. These rules increase the error-proneness and should be re-implemented or removed in future improvements.

- *Abstraction Gradient: types and availability of abstraction mechanisms.*

Abstraction Gradient refers to the level of abstraction presented in a notation. In this case an abstraction is a grouping of elements to be treated as one entity.

The notations of UML elements have been further abstracted to a set of simple geometric shapes, while still organized in a hierarchical structure in our tool. This may raise initial learning curve to use the tool, but in long run provide higher flexibilities and extensibilities to the tool.

- *Secondary notation: secondary notation in means other than formal syntax.*

Secondary notation is the way extra information that can be conveyed to the user via means other than the formal syntax of the language. The escape from formalism refers to the possibility for the user to add information to the model being created in a completely free form way.

Both of them are highly achieved in our tool in that the user has great freedom to sketch whatever secondary notation they desire and to mix notational elements, while conventional UML CASE tools are usually lack of supports in these two aspects.

- *Closeness of mapping: closeness of representation to domain.*

Closeness of mapping refers to how closely the notation or language matches the real world problem it is attempting to model. Ideally each entity in the problem domain would map to a single entity in the notation or language, and operations on those problem entities would similarly be mapped to simple operations on those entities in the language.

In our tool, sketched diagram elements must have a degree of similarity to computer-drawn UML elements in order to be recognized. This constrains the user to a degree and resolving misrecognition can adversely impact on usability.

- *Consistency: similar semantics are expressed in similar syntactic forms.*

Consistency means that the notation/language/tool represents concepts and performs tasks in a consistent manner, i.e. similar things are done in a similar way.

In our tool, as the sketching of UML designs are pen gesture based, the implementation of the manipulations to sketched UML elements are also pen gesture based instead of using other methods e.g. pen-tap on modality buttons etc. to provide consistency to the tool.

- *Diffuseness: verbosity of language.*

A notation is said to be diffuse if it uses many symbols to express a concept, while terse notation use relatively few symbols to convey more information.

In our tool, an almost infinite range of sketched shapes can be recognized as the same UML element due to the use of hand-sketching. This allows users to employ a wider range of symbols than CASE tools with fixed shape computer-drawn models e.g. the user can use size, variations in slope and minor annotations to distinguish elements if desired.

- *Hard mental operations: high demand on cognitive resources.*

Hard mental operations are processes that are made difficult to complete or concepts that are made hard to understand by the language or tool.

Ambiguous sketches cause confusion for our tool and user. The learning curve of text recognition and some complex shape sketching/recognition make learning to use the tool in some respects more difficult to (simple) mouse-driven conventional UML CASE tools.

- *Provisionality: degree of commitment to actions or marks.*

  Progressive evaluation refers to being able to test and evaluate your models during development to get feedback on how the model is developing. It is essentially the ability to test partial systems and models as well as completed ones.

  The shape recognizers and user-demanded formalization of the tool design sketches support progressive evaluation within the tool.

- *Progressive evaluation: work-to-date can be checked at any time*

  Progressive evaluation refers to being able to test and evaluate your models during development to get feedback on how the model is developing. It is essentially the ability to test partial systems and models as well as completed ones.

  The shape recognizers and user-demanded formalization of the tool design sketches support progressive evaluation within the tool.

6.3 User Survey

To assist us in evaluating our work, we carried out a survey of a small group of experienced whiteboard users and UML designers. The survey consists of three parts (see Appendix B for the details):

1) *A brief introduction to our tool and the purpose and contents of the survey* – this provides the basic background information about the tool and the survey to the users.

2) *Two groups of tasks to be performed by the users* – the first is to sketch a number of UML icons following the drawing rules specified in *Appendix A* on both our tool and a conventional whiteboard to assess the reaction speed of our tool (compared with the conventional whiteboard) and the shape recognition accuracy of our tool. The results of the tasks in this group are recorded in a table of the survey to be analyzed. The second is to us our tool to perform part or all of UML designs introduced in the 'A Case Study' chapter. The users are recommended to take notes on these tasks, where the notes will be useful to assist the users answering questions in the questionnaire.

The comparison of the speed to complete UML icon sketching is shown in Table 6.1, and the results of the UML icon recognition accuracy are shown in Table 6.2. We did not carry out any statistical analyses to those results due to our time restriction and the fact of there is only small amount of data collected.

	Our tool	***A conventional whiteboard***
The average time taken for completing the sketching of a UML construct without its text part	5.3 sn	2.6 sn

Table 6.1 Comparison of the speed to complete the sketching of UML icons

From Table 6.1, we can see that the overall average time the user takes to complete a UML icon on our tool is about as twice as the time taken on a conventional whiteboard. We have found that this is due to the reasons of: 1) the delay caused by the MIMIO pen movement capturing system; 2) the drawing rules applied in the tool slows down the sketching speed. We can see from here what can be improved in the future, though the speed of drawing a UML icon in an average 5.3 seconds is felt to be acceptable by most users.

We are very happy with the recognition speed of less than 0.5 milliseconds. This speed is calculated by our tool.

Constructs	***Recognition rate***
Actor	89.3%
Use case	73.2%
Class	88.5%
Object	90.1%
Component	83.0%
Node	69.2%
Note	84.3%
Activation	91.7%
Package	86.1%
Association	89.6%
Dependency	87.3%
Generalization	79.0%
Aggregation	80.2%
Message	88.9%
Average	**84.3%**

Table 6.2 Recognition accuracy of UML constructs

From Table 6.2, we can see that our tool has a high recognition rate with the average score of 84.3%. This indicates that the shape recognition algorithm we implemented in the tool is very powerful. The two items having the highest recognition rates are the Activation (91.7%) and Object (90.1%) icons. The reason for that is because they are two simply rectangles which are easy to drawn. The item with the lowest recognition rate is the Node (69.2%) icon. This can be due to its complexity to be drawn. We may try to find solutions to address this issue in the future. The second worst recognizable icon is the Use case (73.2%). This is bit surprised as it is just a simple Ellipse shape. However, we finally observed that the users are better in drawing straight lines (lines, rectangles, etc.) than curves (circle, ellipse, etc.). This is an interesting finding and we may also try to find solutions to address this issue in the future.

3) *A questionnaire* – the questions contained in the questionnaire mainly focus on getting the feedback on the tool from the users after they have used the tool by

carrying out previous tasks. The feedback should cover aspects of: the proof of our concept; the usability, performance and usefulness of the prototype tool; and the features the tool provided which differ from the conventional whiteboards and conventional UML CASE tools. The questions are not organized under different categories as some of them may cover several aspects. A comments section is provided at the last which expects you write down any comments you have on the tool, where the comments will be a very help for us in reviewing our research and outlining potential future work.

The result from analyzing feedbacks provided by our users indicates the following general characteristics of our work:

- The system is easy to learn and pen manipulations of diagrams provide efficient use of time.
- Good feedback is provided to the users for pen manipulations while in progress and when finished.
- The GUI follows a user friendly design.
- The ability to annotate sketched diagrams in flexible ways is important.
- The tool encourages collaborative UML design.
- The lack of enforcement during sketching of UML diagram constraints encourages exploratory design.
- The drawing rules were annoying for some users and they suggested we would improve this part in our future work.
- The text recognition component was unsatisfactory.

6.4 Our Comments

Reflecting back on our work, we are very happy with things we achieved. Our E-whiteboard UML design tool provides an efficient and effective sketching-based user interface on a large screen E-whiteboard. It greatly supports the nature of hand-sketching in which both design sketch elements and text are constructed and manipulated using pen-based input. The freedom from heavily-enforced modeling constraints and flexible annotation facility in this environment encourages exploratory and collaborative UML-based software design.

We summarize a comparison of differences between our tool and conventional UML CASE tools in Table 6.3, and the differences between our tool and conventional whiteboards in Table 6.4.

	Conventional CASE Tools	***Our Tool***
Input device	Keyboard, mouse	Stylus, E-whiteboard
Output device	Monitor	E-whiteboard
Input method	Keyboard/mouse operation	Pen-based sketching
Secondary notation	Only in UML 'Note' construct	Yes
Color support	Usually not	Yes
Computerized view of UML diagrams	Yes	Yes
Manipulations to UML elements	Yes	Yes
History navigation	Usually yes	Yes
File saving/loading	Yes	Yes
Exporting	Usually	Yes
Support all UML diagrams	A few	Not (current)
Mixture of different UML diagram types	No	Yes
Full UML constraints implementation	Mostly yes	No
Printing	Yes	Yes
Copy and paste to external file format	Yes	No

Table 6.3 General comparison between our tool and conventional UML CASE tools

From Table 6.3, we can see the main differences (advantages) our tool contains compared with conventional UML CASE tools are: 1) the use of E-whiteboard based input/output devices to support the sketching based UML design; 2) the support of the mixture of different UML diagram types; 3) the better support of secondary notations and other facilities like the color. However, conventional UML CASE tools have some strengths over our tool, e.g.1) full implementation of the UML semantics constraints; 2) some special facilities like the copying and pasting diagrams to external file format e.g. the Word document. Some functions have been implemented in both, e.g. manipulation and formulization of UML diagrams, saving/loading and exporting, other facilities like history navigation and printing etc.

	Conventional Whiteboard	***Our Tool***
Input/output method	Physical	Electronic
Availability	High	Low
Cost	Relatively low	Relatively high
Learning curve	Low	Relatively high
Installation	Easy	Relatively hard
Sketching support	Yes	Yes
Free annotation support	Yes	Yes
Speed of completing a UML sketch	Relatively fast	Relatively slow
Color support	Yes	Yes
Drawing restriction (rule applied)	No	Yes
Recognition of drawing	No	Yes
Formalization of UML diagrams	No	Yes
Manipulations to UML elements	No	Yes
History navigation	No	Yes
Sketch saving/reloading	No	Yes
Export to CASE tools	No	Yes
Mixture of different UML diagrams	Yes	Yes
Printing	No	Yes

Table 6.4 General comparison between our tool and conventional whiteboard

From Table 6.4, we can see the main differences (advantages) of our tool compared with conventional whiteboards are: 1) the abilities of recognizing and formalizing UML sketches into standard computer drawn UML diagrams; 2) the support of saving/loading sketched work and exporting formalized UML diagrams to external UML CASE tools; 3) the support of pen action based manipulations of sketches; 4) the support of other facilities e.g. history navigation, printing etc. However, conventional whiteboards still have some advantages which can not be replaced by our tool yet: 1) the high availability and the relatively lower cost; 2) the lower learning curve and easier installation; 3) a faster drawing speed and less drawing rules. Some characteristics of both are similar, e.g. the support of sketching, free annotation and the mixture of different UML diagram types etc.

6.5 Summary

In this chapter, we presented an evaluation of our tool and discussed its strengths and weaknesses compared with the conventional whiteboards and conventional UML CASE tools. This then leads to the final conclusion of our research and the potential future work that will be presented in next chapter.

CHAPTER 7 - Conclusion

7.1 Introduction

In this concluding chapter, we present a summary of the contributions of the thesis to the research field and outline the potential future work. A general summary of the thesis is provided in the end of this chapter.

7.2 Contributions of the Thesis

We feel that this thesis has made several contributions to the research field of developing the E-whiteboard based sketching tool to support the software design. The contributions of the thesis are summarized below.

- *Designed and prototyped the UML sketching tool using the LIDS E-whiteboard technology.*

  This a major part of the work in this research and the main contribution provided by the thesis. The followings are supplemental points we feel are particularly important to this work.

- *The approach of retaining the sketches as long as the user desired while having another view with formalized UML diagrams that can be easily switched to.*

  This approach combined the features from the conventional whiteboard and the conventional UML CASE tools. The retaining of the sketches provides the ‘look and feel’ of a common whiteboard encouraging the expression of design ideas in a natural sketching way without interruptions. The formal view provides the direct feedbacks on their work and a place for checking the UML semantics constraints.

- *The provision of facilities for saving/loading sketches as well as exporting formalized UML diagrams using the XML technology.*

  This is an effective implementation method which provides an advantage of the efficiency to the tool over the conventional whiteboard. It also establishes a communication manner between our sketching tool and conventional UML CASE tools which can be taken into a further investigation on the issue of integrating of our tool with other UML CASE tools.

- *The investigation of the refined multi-stroke shape recognition algorithm.*

  The refined multi-stroke recognition algorithm has been proved as an efficient method for recognizing UML notion shapes. It is more suitable to support the sketching of UML notations in a natural way (multi-stroke) than other single-stroke based recognition algorithms, and yet more acceptable by the users as less work is required (compare with the single-stroke algorithm where a considerable amount of trainings are required).

- *The mechanism of abstracting UML notations into the combinations of simple geometric shapes.*

  This mechanism implemented in our tool is in conjunction with the use of the multi-stroke shape recognition algorithm described above. We feel that this mechanism added a high extensibility to the tool as we assume that the majority of notation based modeling tools including those in the fields other than the software design can apply this abstraction mechanism. Thus the refined multi-stroke shape recognition algorithm can be potentially adopted to support sketching the design models in many application areas.

- *The implementation of the drawing-rule based filtering technique used in the multi-stroke recognition algorithm.*

Although this technique is sometimes considered by the users as a drawback of the tool, we argue that it provides an alternative method with a main advantage of enhance the performance of the recognition by adding a ‘little’ amount of well designed drawing rules with a very low learning curve. This issue can be further investigated.

- *The implementation of the pen interaction based sketch manipulation technique.*

  This pen interaction based (rather than use other methods e.g. click buttons) sketch manipulation technique adds another extra function to the tool over the conventional whiteboard while still keep the characteristics of working with the conventional whiteboard – using the pen as much as possible.

7.3 Future Work

The future work outlined here are the parts in our tool that we feel need to be improved or extended, and the possible further investigations/applications which we see as the consequences of our work.

- *To re-develop the text recognition mechanism.*

  As we have stated out that the text recognition mechanism used in our tool is not ideal and is only a temporary solution, we plan to re-develop this part by seeking a more efficient and sophisticated technology to address this problem. A third parties’ component like the Microsoft’ Tablet® PC Toolkit may be a good start of the investigation.

- *To re-investigate the ‘Drawing-Rule Based Filtering’ technique used in the refined multi-stroke shape recognition algorithm.*

  We have pointed out that this technique is an arguable part in our work which may be good or may be bad. We plan to take this issue to a further investigation.

- *To develop a generic editing component for the user to define abstractions of the modeling notations.*

  We have developed in our tool the mechanism of abstraction of notations into combinations of simple geometric shapes for the sketch recognition. However, this pat is currently 'hard coded' in the program particularly tailed for working with the UML notations only. This is then lack of efficiency and flexibility. We plan to develop a generic editing component which allows the user to define the abstractions them. For example, to select a 'circle' icon (provided by the tool) and then a 'horizontal line' and then a 'vertical line' and then two 'oblique lines' to the editing area with each of them in a proper poison and size to define the 'Actor' icon of the UML notation. This will potentially work for other modeling languages and thus provide the flexibility to the tool.

- *To investigate a suitable generic XML based protocol as a communication channel between our tool and other UML CASE tools.*

  We have used the XML based technology in our tool to perform the task of exporting UML designs to other UML CASE tools. However, it is not a generic model to be supported by other tools yet. We plan to investigate on this issue to design an XML based protocol that is suitable for encoding and transforming the UML designs.

- *To develop a distributed E-whiteboard system to support collaboratively utilizing multiple e-whiteboards.*

  This can be seen as a more challenging project, which upon its success will enable multiple E-whiteboards to be used collaboratively among designers from distances (different rooms, buildings or cities).

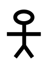
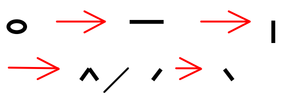
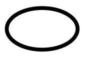
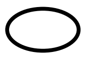
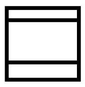
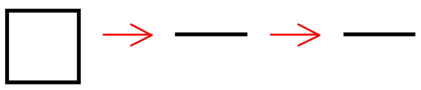
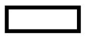
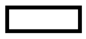
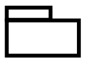
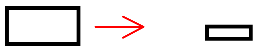
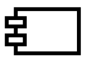
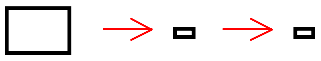
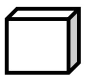
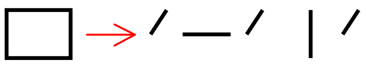
- *To investigate the use of our application in other pen based computing systems.*

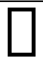
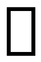
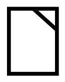
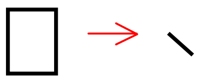
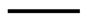
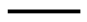
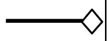
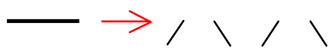
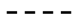
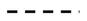
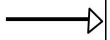
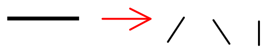
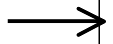
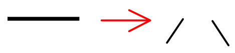
Potentially our application can be used in other pen-based computing systems like the Tablet PC mentioned previously. However, there will be different issues need to be investigated.

7.4 General Summary

In this thesis, we have investigated issues involved in developing an E-whiteboard based sketching tool to support early-stage UML design. We first stated out our motivation and goal for initiating this research. After that, we provided a comprehensive overview of the background study we have carried out including the basic concepts and technologies for developing such a tool and other research related to this area. Then, based on our goal, the background study and a survey form potential users, we specified the key requirements of the functionality the tool should provide. Next we discussed in detail the issues involved in the design and implementation of the tool. A short case study was provided to further illustrate the use of the prototyped tool. Finally we presented an evaluation of the tool where we argued that our E-whiteboard based UML sketching tool has unique features and advantages over the conventional whiteboard as well as the conventional UML CASE tools.

Appendix A - Drawing Rules for Sketching UML Icons

Icon	Drawing order	Position, size and other notes
		1. each of the left shapes must be in appropriate position 2. the horizontal line must be in approximately the half of the high of the sketch, and its width is the width of the whole sketch 3. the vertical line must be in approximately the half of the width of the sketch
		1. The ellipse can be drawn either clockwise or counter-clockwise in one stroke
		1. the rectangle can be drawn by multi-strokes in any order 2. the upper horizontal inner line must be drawn in the second last order, and within the upper one out off three area, and not outside the boundary of the sketch, and the width is not lass the width of the sketch too much (within 10 pixel) 3. the lower horizontal inner line must be drawn in the last order, and within the lower one out off three area, and not outside the boundary of the sketch, and the width is not lass the width of the sketch too much (within 10 pixel)
		1. the rectangle can be drawn by multi-strokes in any order, its width must be more than two times greater than its height
		1. the lower larger rectangle can be drawn by multi-strokes in any order 2. the upper smaller rectangle must be drawn in the last, and must be drawn in a single-stroke, and its left edge is as same as the left edge of the larger rectangle, and its bottom touches the top of the larger rectangle, and its height is less than the half of the high of the larger rectangle, and its width is less than the two out off three of the width of the larger rectangle
		1. the largest rectangle can be drawn by multi-strokes in any order 2. the upper smaller rectangle must be drawn in the second last, and must be drawn in a single-stroke, and its width is less than the half of the width of the larger rectangle, and its height is less than one out off four of the height of the height of the larger rectangle, and its vertical position is at approximate upper one third of the height of the larger rectangle 3. the lower smaller rectangle must be drawn in the second last, and must be drawn in a single-stroke, and its width is less than the half of the width of the larger rectangle and its height is less than one out off four of the height of the height of the larger rectangle, and its vertical position is at approximate lower one third of the height of the larger rectangle 4. the left edge of the larger rectangle must pass through approximately the middle of each smaller rectangles
		1. the rectangle must be drawn first by multi-strokes in any order 2. the rest one horizontal, one vertical and three oblique lines can be draw in any order but must be in one stroke and at proper positions, these position are not described , but they are easy to understand if refer to the left diagram

		1. the rectangle can be drawn by multi-strokes in any order, its height must be more than two times greater than its width
		1. the rectangle can be drawn by multi-strokes in any order 2. the oblique line is drawn at last at the upper corner of the rectangle, its upper end is within the left half of the rectangle, its lower end is at the upper half of the rectangle
		1. the line is drawn in one stroke
		1 the longer line must be drawn in one strokes first 2 the four short lines can be drawn in any order but each in one stroke and at proper position (please refer to the left diagram for the positions)
		1 the line is drawn in multi-strokes (dashed)
		1 the longer line must be drawn in one stroke first 2 the three short lines can be drawn in any order but each in one stroke and at proper position to form a triangle (please refer to the left diagram for the positions)
		1. the longer line must be drawn in one stroke first 2. the two short lines can be drawn in any order but each in one stroke and at proper position (please refer to the left diagram for the positions)

- Appendix B - User Survey of an E-whiteboard Based UML Design Sketching Tool

1 Introduction

First, we give a brief introduction to our tool to be evaluated in this survey. If you have already known our tool very well, you may skip this section.

The prototype tool we have developed in our recent research is based on the E-whiteboard technology to support early stage sketching of UML design diagrams. It uses a large E-whiteboard and an inkless-pen (stylus) as the main input/output devices. Users draw UML diagrams and secondary annotations of the software design on the screen of the E-whiteboard using the stylus. The tool reads in the pen-based input and generates the computer drawn virtual ‘ink’ which is projected on the screen by a projector as the drawing mark of the stylus. The sketches are retained on the screen after they have been drawn. Manipulations to the sketches (to move, copy, replace and delete sketches) can be performed by the pen-based actions. Text entry areas (indicated as dotted pink lines) are added to inform the successes of the recognition, and those recognized UML sketches are formalized into computer drawn standard UML diagrams on the background process and the formal view can be displayed when the user desired. The sketches can be saved and loaded to and from files. The formalized UML diagrams can be exported to a conventional UML CASE tool.

2 Survey Overview

This survey is undertaken to fulfill the requirement of evaluating the tool introduced above. It contains two major parts: 1) to use our tool by performing part or all of the tasks which are described in detail below; 2) to complete the questionnaire.

The survey is expected to take approximately 1~3 hours depending on how many tasks and questions you will take and answer. When you are ready for the survey, you will be guided by one of us to the place where the tool will be already set up with the application running on it.

3 Tasks

There are two groups of tasks. The guider from us will give you a short demonstration on how to use the tool before performing each of the tasks.

3.1 Task group one

1. Read the provided document of the 'Drawing Rules for Sketching UML Icons in Our Tool' with the drawing rules you are asked to follow when sketching.

2. Sketch 5~10 of each UML icon as on the most left column of the table in the drawing rule document. If you do not have time to sketch all those icons, you may sketch some of them of your choices. Please record the total number and the total drawing time of each sketched UML icon, and the recognition rate for each sketched UML icon in Table 1.

3. Sketch 5~10 of each UML icon as on the most left column of the table in the drawing rule document. Please do not sketch text for the name/properties/methods of those recognized UML sketches here. If you do not have time to sketch all those icons, you may sketch some of them of your choices. Please record the total number and the total drawing time of each sketched UML icon, and the recognition rate for each sketched UML icon in Table 1.

4. Sketch same UML icons on a provided conventional whiteboard. Please record the total number and the total drawing time of each sketched UML icon in Table A-1.

	Use our tool			*Use a whiteboard*	
UML Icon	**Total no. of sketches**	**Total sketching time (min)**	**No. of recognized sketches**	**Total no. of sketches**	**Total sketching time (min)**
Actor					
Use case					
Class					
Object					
Component					
Node					
Note					
Activation					
Package					
Association					
Dependency					
Generalization					
Aggregation					
Message					

Table A-1. Information recorded from task group one

3.2 Task two

1. Read the provided document of the 'Case Study'.

2. Sketch part or all of the three UML diagrams contained in the document, the *Use case diagram*; the *Class diagram*; and the *Sequence diagram*.

3. Perform some or all of the manipulations to any sketches you have drawn. This will be guided by the demonstrator from us.

4. No information is required to be recorded in a formal document. However, we recommend you to take your own notes regarding issues of the usability, performance, usefulness of the tool; the differences of it comparing with conventional whiteboards as well as conventional UML CASE tools based on you experiences with them; and any other comments you would like to make. The notes will be very useful to assist you in answering questions in the following questionnaire.

4 Questionnaire

We would appreciate if you would take the time to complete this questionnaire. The questions in this questionnaire mainly focus on getting the feedback on the tool from you after you have used the tool by carrying out previous tasks. The feedback should cover aspects of, the proof of our concept; the usability, performance and usefulness of the prototype tool; and the features the tool provided which differ from the conventional whiteboards and conventional UML CASE tools. The questions are not organized under different categories as some of them may cover several aspects. We provide a comments section at the last expecting you to write down any comments you have on the tool, where your comments will always be a great help for us in reviewing our research and outlining potential future work.

Do you think our concept of developing this kind of tool is in some value? Why?

Is it easy or difficult to learn to use the tool? Why?

Is it easy or difficult to use the tool for sketching UML diagrams? Why

Do you think it is a good approach to retain sketches after they have been drawn? Why?

Do you think the tool provides good feedbacks on the recognition results to the user? Why?

What do you think about the formalization facility supported by the tool and the provision of a formal view? Why?

Do you think the support of the secondary annotation is important? Why?

What do you think about the pen action based manipulation approach in the tool? Why?

What do you think about those ‘drawing rules’? Why?

What do think about the overall performance of the tool?

What does the tool do well? Why?

What does not the tool do well? Why?

What are the main features the tool has comparing with the conventional whiteboards?

What are the main features the tool has comparing with the conventional UML CASE tools?

Are there any improvements can be made to the tool in your opinion?

Which one would like to use for doing the celebrative UML design when both our tool and a conventional whiteboard are available to you? Why?

Which one would like to use for doing the celebrative UML design when both our tool and a conventional UML CASE tool are available to you? Why?

Are there any other comments you would like to make?

Bibliography

[Apte et al 93] A. Apte, V. Vo, T.D. Kimura, “Recognizing Multistroke Geometric Shapes: An Experimental Evaluation”, Proceedings of the 6th annual ACM symposium (1993) on User interface software and technology, ACM Press, pp. 121-128. (1993)

[Apperley et al 02] M. D. Apperley, B. G. Dahlberg, A. Y. Jeffries, L. B. Paine, M. Phillips, W. J. Rogers, “Lightweight capture of presentations for review”, Proceedings of IHM-HCI, Lille, France, ACM Press, (2002)

[Bailey et al 03] B. P. Bailey, J. A, Konstan, “Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design”, Paper/short Talks of CHI 2003 on Software Development, ACM Press, pp. 313-320. (2003)

[Berque et al 02] D. Berque, D.K. Johnson, L. Jovanovic, “Teaching theory of computation using pen-based computers and an electronic whiteboard”, ACM SIGCSE. Bulletin, vol. 33, no. 3, September 2001, ACM Press, pp.169-172.

[Chatty et al 99] S. Chatty, P, Lecoanet, “Pen Computing for Air Traffic Control”, Proceedings of CHI 2000, ACM Press, pp. 51-65. (1999)

[Chen et al 03] Q. Chen, J. Grundy, J. Hosking, “An E-whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams”, Proceedings of the VMSE2003: IEEE symposium on Visual/Multimedia Software Engineering (to be published on October 2003).

[Damm et al 00] C. H. Damm, K. M. Hansen, M. Thomsen, “Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard”, Proceedings of CHI 2000 on Human factors in computer systems: the future is here, ACM Press, pp. 518-525. (2000)

[Fowler 97]	M Fowler, “UML Distilled”, Addison Wesley (1997).
[Green et al 96]	T.R.G. Green, M. Petre, “Usability analysis of visual programming environments: a ‘cognitive dimensions’ framework”, Journal of Visual Languages and Computing 1996 (7), pp. 131-174.
[Gross et al 96]	M. D. Gross, E. Y, Do, “Demonstrating the Electronic Cocktail Napkin: a paper-like interface for early design”, Proceedings of CHI 1996, ACM Press, pp. 32-37. (1996)
[Grundy 00]	J. Grundy, “Requirements Engineering & OOA Tutorial; Software Architecture & Object-oriented Design Tutorial”, Tutorials of the Software Engineering lecture at the Department of Computer Science of The University of Auckland. (2000)
[Guimbretiere et al 01]	F. Guimbretiere, M. Stone, T. Winograd, “Fluid Interaction with High-resolution Wall-size Displays”, Proceedings of UIST 2001, ACM Press, pp. 21-30. (2001)
[Igarashi et al 00]	T. Igarashi, W. K, Edwards, A. LaMarca, E. D. Mynatt, “An Architecture for Pen-based Interaction on Electronic Whiteboards”, Proceedings of AVI 2000, ACM Press, pp. 68-75. (2000)
[Landay 96]	J. A. Landay, “SILK: sketching interfaces like krazy”, Proceedings of CHI’96 on Human factors in computer systems: common ground, ACM Press, pp. 518-525. (1996)
[Landay et al 96]	M. D. Gross, E. Y, Do, “Demonstrating the Electronic Cocktail Napkin: a paper-like interface for early design”, Proceedings of CHI 1996, ACM Press, pp. 32-37. (1996)
[Lank et al 01]	E. Lank, J. Thorley, S. Chen, D. Blostein, “On-line recognition of UML diagrams”, Proceedings of the Sixth International Conference on Document Analysis and Recognition, IEEE CS Press, 2001, pp.356-360. (2001)

[Microsoft 03]	Microsoft® home page: available from http://www.microsoft.com (Last Visited Friday, July 11, 2003)
[MIMIO 03]	MIMIO® home page: available from http://www.mimio.com (Last Visited Friday, July 11, 2003)
[Myers 98]	B.A. Myers, "A Bief History of Human-Computer Interaction Technology", Journal of Interaction, March – April, 1998, pp. 44-54. (1998)
[OCHRE 03]	OCHRE – Optical Character Recognition Using Neural Networks in Java available from http://www.geocities.com/siliconvalley/2548/ochre.html (Last Visited Friday, July 11, 2003)
[Perlimmer et al 02]	B. Perlimmer, M. Apperley, "Computer-aided sketching to capture preliminary design", Proceedings of the Third Australasian Conference on User interfaces, Australian Computer Society, Inc, pp. 9-12. (2002)
[Rational 03]	Rational Software Coperation home page: available from http://www.retional.com (Last Visited Friday, July 11, 2003)
[Riordan 00]	R. Riordan, "Designing Relational Database Systems", Microsoft Press. (2000)
[Rubine 96]	D. Rubine, "Specifying Gesture by Examples", Proceedings of the 18th annual conference(1991) on Computer graphics and interactive techniques, ACM Press, pp. 329-337. (1991)
[Rumbaugh et al 91]	J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice Hall (1991).
[SMART 03]	SMART® home page: available from http://www.smarttech.com (Last Visited Friday, July 11, 2003)

[UML 03]	UML™ Home Page, OMG (Object Management Group), available from http://www.uml.org/ (Last Visited Friday, July 11, 2003)
[UML Center 03]	UML Center: available from http://www.smartdraw.com/resources/centers/uml/uml/htm (Last Visited Friday, July 11, 2003)
[UML Tools 03]	UML Tools: available from http://www.jeckle.de/umltools.html (Last Visited Friday, July 11, 2003)
[Voida et al 02]	S. Voida, G.M.Corso, E.D.Mynatt, B. MacInt, "Integrating virtual and physical context to support knowledge workers", IEEE Pervasive Computing, vol. 1, no. 3, July-Sept. 2002, IEEE CS Press, pp.73-79.
[XML 03]	Extensible Markup Language (XML) available from http://www.w3.org/xml (Last Visited Friday, July 11, 2003)
[Zhao 93]	R. Zhao, "Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors", Proceedings of CHI 1993, ACM Press, pp. 95-100. (1993)