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

Qi Chen[1], John Grundy[1, 2] and John Hosking[1]

Department of Computer Science[1] and Department of Electrical and Electronic Engineering[2],
University of Auckland, Private Bag 920, Auckland, New Zealand
{qche013@ec.|john-g@cs.|john@cs.}auckland.ac.nz

Abstract

We describe a Unified Modelling Language (UML) diagramming tool that uses an E-whiteboard, pen-based sketching interface to support collaborative design. Our tool allows designers to sketch UML visual modelling language constructs, mixing different UML diagram components, free-hand annotations and hand-written text. A key novelty of our approach is the preservation of hand-drawn diagrams and support for manipulation of the diagrams using pen-based actions. UML sketches can be "formalized" to computer-recognised and drawn diagrams, and exported to a 3rd party CASE tool.

1. Introduction

One of the most common tools used by software designers when doing collaborative design work is a whiteboard. This is used to collaboratively sketch software design ideas (for example as whole or partial UML diagrams), explore architectural solutions, capture high level code fragments, organise design teams, schedule events, etc, as shown in Figure 1 [5, 11].

Three partial UML diagram types are shown in the whiteboard sketches on the right – (1) "use cases" (stick figure and oval), describing actors (users) interacting with a system; (2) "classes" (box with horizontal lines inside and arrowed lines between), describing classes of types and their relationships; and (3) "sequences" (boxes with vertical lines underneath and horizontal arrowed lines between), denoting message sequence flow between objects. Some key advantages of using 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, delete or extend 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 colour. Sketches do not have to be precise nor complete in any formal manner.
- 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

Disadvantages of conventional whiteboards for such tasks are a lack of data persistency, an inability to readily transfer information to electronic design tools (eg CASE tools), difficulty making some changes (eg repositioning parts of diagrams), lack of collaboration support at a distance, and ink dust on your clothing. For these reasons much recent work has focussed on the development and use of large electronic whiteboards [19, 15, 3, 11].

We describe an electronic whiteboard-based early design phase sketching tool. This allows UML diagrams to be sketched, recognised, and integrated with a conventional CASE tool. Key novelties of our approach include the preservation of hand-sketched design elements, provision of various pen-based manipulation facilities on sketches, and ability to formalise sketches to computer-drawn diagrams for export to CASE tools.

2. Related Work

Electronic whiteboards have become a popular way of support a wide range of activities. These include meeting support with collaborative document display, review and annotation [21]; education [3]; presentation control and annotation [1]; and (early) design [11]. The previously listed 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 [1, 3, 19].

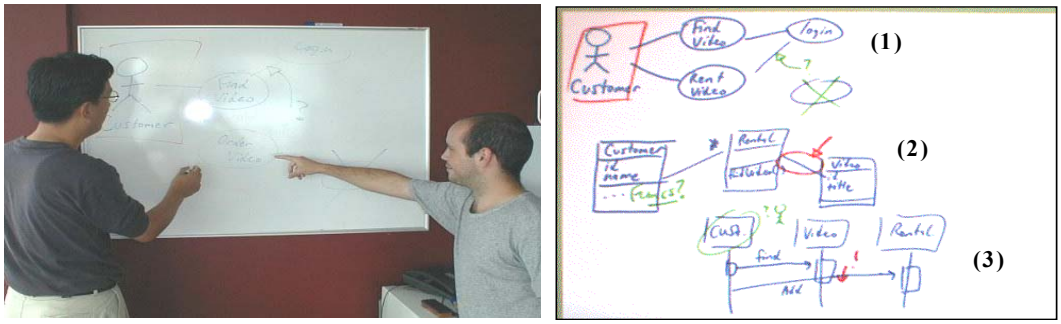

Figure 1. (a) Designers around a white board; (b) an example of sketched UML diagrams.

The Unified Modelling Language [6, 20] has become a standard visual modelling language for software specification and design. UML provides several diagram types, each with more or less independent notations for specific design tasks e.g. use case diagrams for describing system interaction, class diagrams for static object type structures, sequence diagrams for dynamic message flow between objects. Many computer-aided software engineering (CASE) tools have been developed to support UML modelling, with almost all of these tools adopting conventional mouse and keyboard input and standard monitor display of information [16, 17]. However, many HCI studies have pointed out that developers prefer sketching designs by hand rather than using a keyboard/mouse and a computer screen, especially in the early stage of software design.[5, 10, 15]. Empirical studies of CASE tool usage have show that designers find these overly restrictive during early design, often leading to poor utilisation of the tools [9, 11, 5].

A number of systems have been developed using pen-based interfaces to support a sketch-based approach to software design. One of the earliest was SILK [10] which allows software designers to sketch an interface using an electronic pad and stylus. SILK attempts to recognize user interface widgets and other interface elements as soon as they are drawn, though it is not intrusive and users are only made aware of the results when they choose to exercise the widgets. The recognition uses Rubine's single stroke gesture recognition algorithm [18]. When the designer is satisfied with the early prototype, SILK can transform the sketches into standard Motif widgets and graphical objects. At each stage of the process, the designer can switch the sketch into run mode to test the interface by manipulating it with the mouse, keyboard, or stylus. SILK stores the history of all drawings for later use. Annotation and editing are also supported in the tool. Silk only recognizes a few ways of drawing each widget and does not support specification of widget behaviour.

FreeForm [15] is a Visual Basic (VB) Add-In for the design of VB forms. It adopts the same metaphor of the SILK , but uses an electronic whiteboard and pen input to support hand-drawn sketching. There are five major parts to the software: the sketch space, storyboard, run mode, recognition engine and VB form converter. 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 [18] algorithm with shape/letter library and rule mapping techniques. Plimmer discovered that retention of the sketch based format while doing user testing improved the quality of testing over conversion to VB forms and hence the quality of the software design.

Knight [5] is the work closest to our research, although other Knight-style UML tools exist [11, 8]. Knight supports collaborative UML modelling using gestures on an electronic whiteboard with pen input. To achieve intuitive interaction, Knight uses compound gestures and eager recognition. Compound gestures combine gestures that are either close in time or space to form 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 by normal keyboard, on-screen Virtual Keyboard, Stylus-based Gestures (as on PDA's), Cirrin, and Quikwrite. Most UML sketching tools like Knight adopt immediate recognition and computer-drawing of information. However, to support informality incomplete elements can be recognized at a later time and a separate "freehand" mode can be used for arbitrary sketches and annotations. However, there is no association between a "freehand" element and a "formal" (recognized) element. Other related work includes work done on recognition of UML shapes from glyphs [11], web interface design in Denim [12], and gesture-based document manipulation [1, 14].

3. Our Approach

Our primary motivation in developing a new, E-whiteboard, sketching-based UML design tool was to

explore the retention of the hand-drawn sketch "look and feel" of real whiteboards with UML sketches, while retaining the ability to recognise and convert the sketches to more formal diagrams. In particular we are influenced by Plimmer's observations in her Freeform work that retaining a sketch form encourages more experimentation with design. Our approach, therefore, is an electronic-whiteboard based sketching tool that recognises UML constructs as they are drawn.

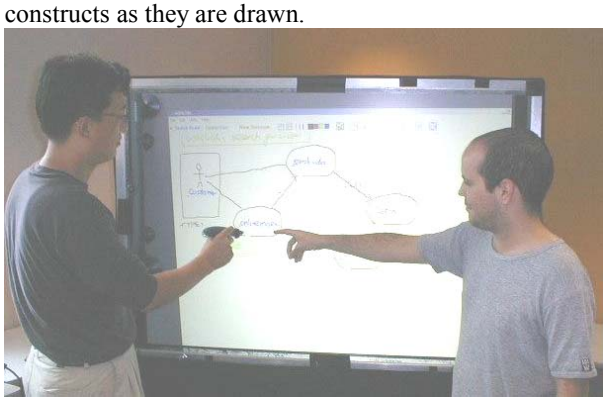

Figure 2. Our SUMLOW design tool in use.

This is very different to Knight and other approaches taken to UML sketching to date [5, 8, 11]. In our tool the look and feel of the hand-drawn constructs are retained as much as possible, while still allowing constructs to be moved, copied, replaced, deleted, etc via pen-based input techniques. Rich, user-defined secondary notation is supported in a seamless way by use of textual annotations, sketch constructs that are not recognisable as UML constructs, colour, etc. Our tool is also unusual in that we allow constructs from different UML diagram types to be mixed together in ways that may violate a particular diagram semantics, but which may be of value during conceptual design. Diagrams or parts of diagrams can be progressively formalised when desired then checked for feedback on semantic constraints, and exported to a standard UML design tool for further work.

Figure 2 shows our system, SUMLOW (Sketched UML On Whiteboard), in use. The electronic whiteboard we have used is a LIDS (Large Image Display Surface) unit [1] which has a large backlit display (eliminating shadowing) combined with a Mimio [13] ultrasonic system for pen location, but the approach is potentially applicable to other pen based systems. The SUMLOW application is implemented using Visual Basic.

Figure 3 shows two screen dumps from SUMLOW in use. Figure 3 (a) shows SUMLOW's sketch board, where several UML constructs, together with some annotations (text and arrow), have been sketched. Constructs can be moved and copied; the right hand class construct was copied from the left one. Note that dotted text entry lines have been added to constructs that have been recognised to indicate where to add names, attribute information, etc. Figure 3 (b) shows SUMLOW's diagram view which displays the recognised shapes in formalised form. Note that in this SUMLOW view the annotations have been omitted as they are not formally recognised constructs. When exporting such formalised UML diagrams to a CASE tool, diagram elements belonging to particular UML diagram types are filtered and included in multiple UML diagrams in the tool.

A Time-out technique is used to process pen input for manipulating diagrams, whereby if a pen is rested on a component for a brief period, this indicates a pen operation, such as moving the construct, is to be undertaken. Single gesture recognition, using Rubine's algorithm, is used for text recognition. Multiple gesture recognition is used to recognise shapes.

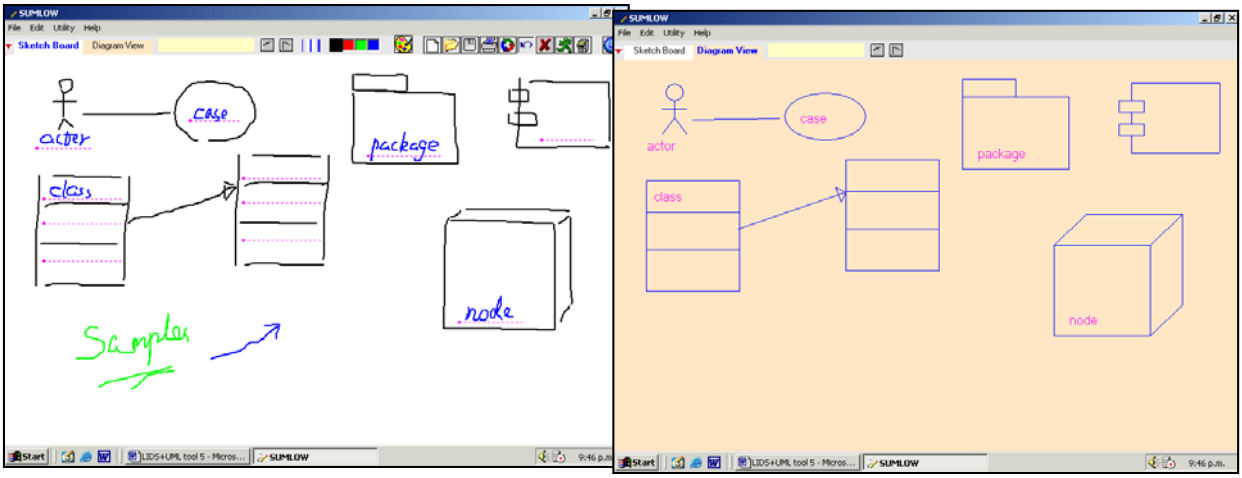

Figure 3. SUMLOW in use illustrating various recognised UML constructs (a) sketch view (b) diagram view.

4. An Example

To illustrate use of SUMLOW we describe the collaborative design of a simple on-line video rental system for a video store. Designers John and Michael use SUMLOW to develop early-phase UML designs together and then formalise their designs and export them to a CASE tool, to support detailed design and system implementation.

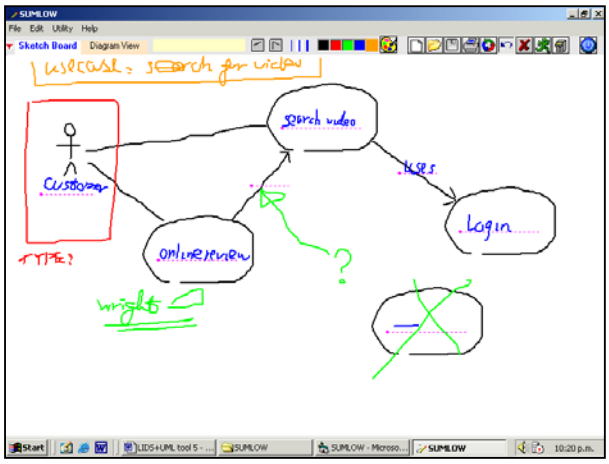

Figure 4. Use case model sketch in SUMLOW.

John begins by using SUMLOW to sketch out the main "use cases" (groups of user-system interaction requirements) using UML use case diagram elements. He draws on the E-whiteboard surface with a stylus (pen without ink) and SUMLOW draws connected pixels as John moves the stylus. John draws one shape after another ("Actors", which are stick-figures and "use cases", which are named ovals), connecting them with interaction relationships.

Shapes are recognised by the multi-stroke gesture recognition algorithm as the designer makes changes to the sketch . Once the shape is identified as an actor or use case, this is recorded and a text entry area (dotted line) is added for entering the construct's name. Unrecognised sketches become secondary diagram annotations.

Shapes are manipulated with pen gestures to indicate movement, and deletion. SUMLOW carries out a simple redrawing algorithm to redraw sketched connector lines between shapes. Figure 4 shows the resulting use case diagram in SUMLOW, with some custom annotations. 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.

After John has sketched out these use cases, Michael takes over to sketch out some initial classes (object types) and 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). Our multi-stroke recognition algorithm is used to recognise these and add three text entry areas to the sketched shape, one for each kind of text item the user can draw. Figure 5 shows his first class. As Michael writes names for attributes and operations on the attribute name and operation name text entry lines in SUMLOW, 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. 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 transferred across to the new shape, as shown in the bottom view in Figure 5.

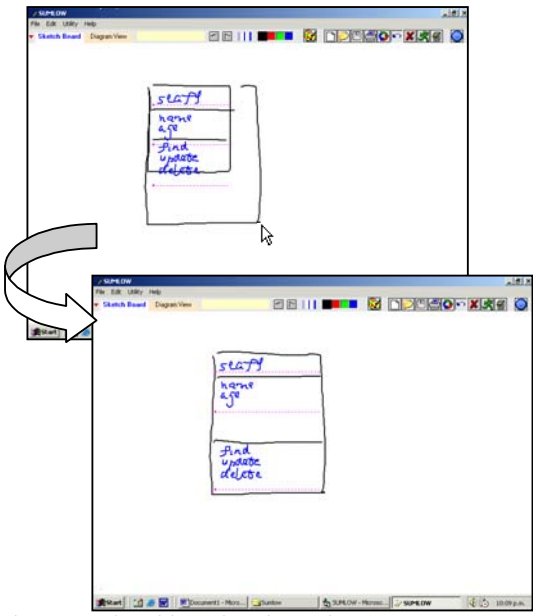

Figure 5. Sketching class icons in SUMLOW.

A more complete UML class diagram sketch is shown in Figure 6, 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 annotation). During design he and John have also added textual annotation, arrows, and shape highlights which are not recognised as UML constructs and hence regarded as secondary notation.

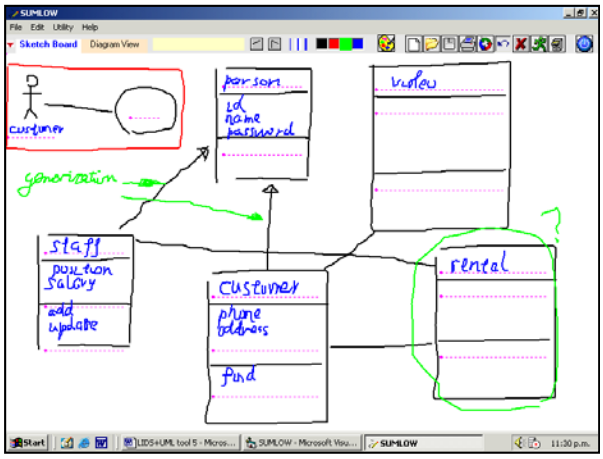

Figure 6. UML class diagram sketch in SUMLOW.

After discussing and modifying the initial class diagram sketch in SUMLOW, John and Michael focus on one of the complex message flows in the proposed video system design. They sketch a UML sequence diagram in SUMLOW to try and capture and discuss this dynamic system behaviour. Figure 7 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 UML diagramming convention, but is here useful for John and Michael in discussing their design sketch.

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. At that point, any other existing sketches on the whiteboard are saved or discarded by user choice, and the horizontal line converted to a solid blue line. Actors or objects drawn in the sketch board will be relocated at the top of the sequence diagram and a timeline (dotted blue line) added associated with that component. Calls and timing elements are sketched on these timelines. Copying, moving or deleting an actor or object will also reposition the timelines, calls, and timing elements as appropriate.

As John and Michael perform their design sketching on the sketch board sketches are formalised in a background process and rendered into formalised UML diagrams in the design view. The results for some of these sketches are shown in Figure 8. Note that some information is discarded from the sketches e.g. informal secondary notation like highlights that have no UML notation equivalent.

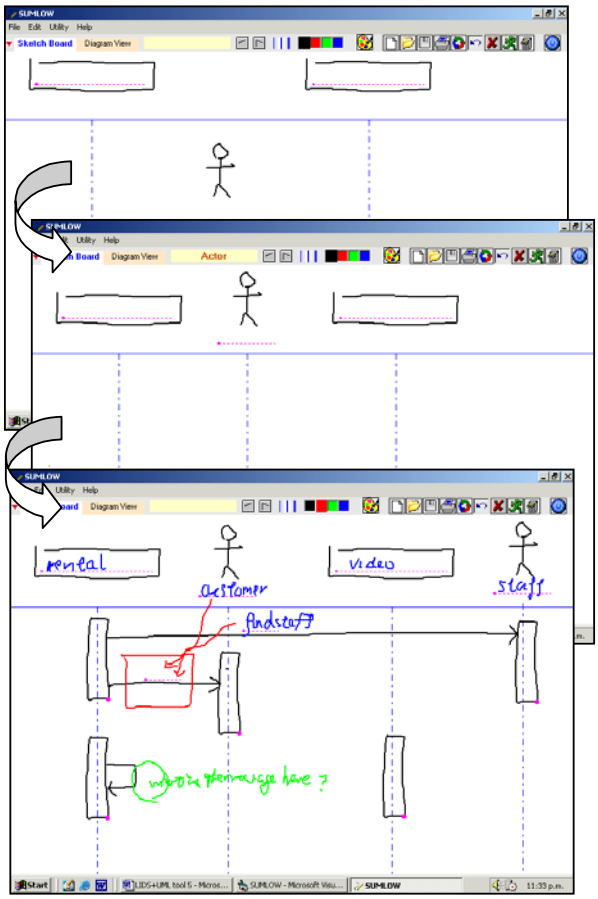

Figure 7. UML sequence diagram sketching.

The two views are completely integrated; except for drawing new objects in the diagram view, objects in the diagram view can be moved, copied and deleted and the manipulations will be reflected in the sketch views. The formalised UML diagrams can be exported to a 3rd party CASE tool using an XML-based design model encoding XMI.

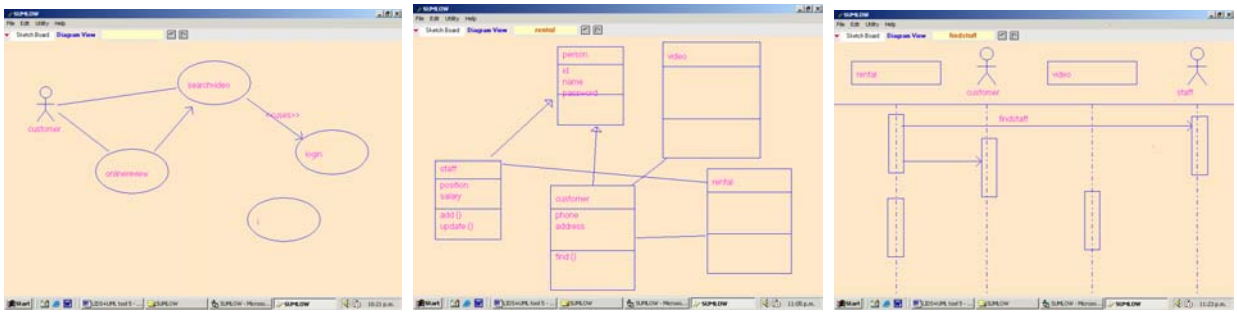

Figure 8. "Formalised' UML diagrams from previously illustrated sketches in SUMLOW.

5. Design and Implementation

Figure 9 shows the basic components of our SUMLOW UML E-whiteboard design tool. The LIDS E-whiteboard provides a data projector, for displaying the SUMLOW user interface, that is back-projected onto an opaque surface. A Mimio data capture device provides pen input for the application. The SUMLOW application is written in VisualBasic and uses VB user interface libraries to provide the sketching interface and application management. The MimioMouse application is used to convert the pen input into simulated mouse movements, used to drive the application's VB controls. This allows conventional VB UI controls to be used but also provides fine-grained sketching tool support via the Mimio stylus pen, used for most of the sketch manipulation. One disadvantage of this input approach is that no right-mouse button is supported, limiting some interaction styles in our interface design (e.g. use of pen-tap on modality buttons rather than in-context pop-up menus).

The multi-stroke algorithm for shape recognition, we adapted from Apte et al. [2], has the advantages that more complex shapes can be handled than with a single stroke algorithm, drawings are more natural, and no training is required, at the expense of being able to recognise only combinations of simple geometric shapes and the need to order appropriately the way in which complex shapes are put together. Recognition is very efficient with all shapes recognised within 2 milli-seconds.

As mentioned previously, Rubine's single stroke algorithm has been used so far in SUMLOW for text recognition. This is not ideal, due to the need for considerable per-user training to provide acceptable recognition accuracy. We see this algorithm as an interim solution until more robust algorithms, such as that used by Microsoft's Tablet PC extensions, have readily accessible APIs that we can make use of easily from SUMLOW.

SUMLOW design sketches are saved and loaded using custom XML encodings. SUMLOW also supports saving formalised UML diagrams to XMI encodings for export to 3rd party CASE tool usage. We do not currently support the import of XMI-encoded UML diagrams from CASE tools. However, this might be provided in future to allow the use of SUMLOW for discussion, annotation and modification of existing UML designs from within the E-whiteboard design environment.

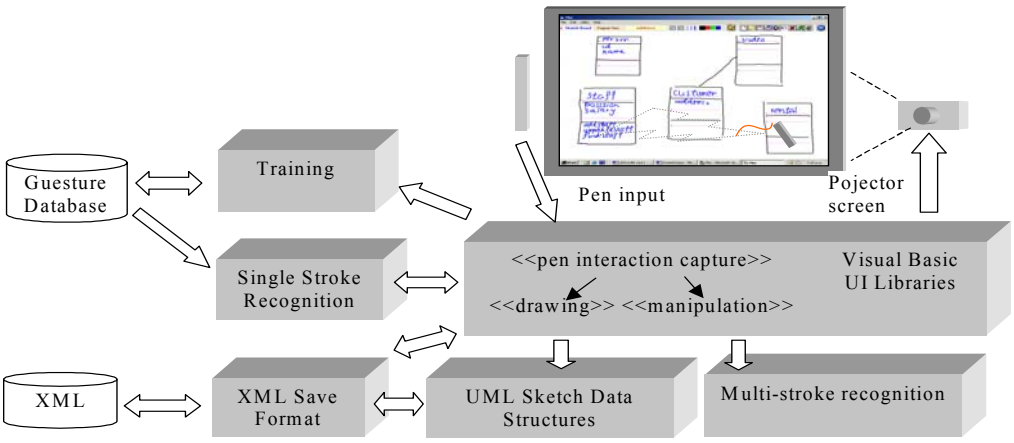

Figure 9. An overview of our SUMLOW tool's architecture.

6. Discussion

We have carried out two evaluations of SUMLOW: one a survey of six experienced UML designers and whiteboard users and to gain subjective feedback on the tools suitability for UML-based software design. The second a Cognitive Dimensions [7] evaluation of SUMLOW to gauge its performance characteristics compared to conventional UML design tools.

Our end-user evaluation did not address the text recognition component, due to the known deficiencies in the approach taken, and concentrated instead on 1) assessing the accuracy of the shape recognition component and 2) utility for UML diagram production. Results of the accuracy evaluation are shown in Table 1.

Constructs	Recognition rate
Actor	100%
Use case (after actor drawn)	80%
Use case (alone)	70%
Class	100%
Object	100%
Component	90%
Node	80%
Note	90%
Activation	100%
Package	100%
Association	100%
Dependency	100%
Generalization	100%
Aggregation	80%
Message	90%

Table 1. Accuracy of SUMLOW shape recognition.

This analysis shows acceptable performance for most components. Some diagram types are more distinctive than others, resulting in higher recognition rates. For example, use cases being simple ovals are more difficult to accurately recognise than classes and components, which are quite distinctive.

Results of the usability evaluation are still being analysed, but preliminary anecdotal feedback provided by our users indicates the following general characteristics of SUMLOW:

- The system is easy to learn and pen manipulations of diagrams provides 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 text recognition component was unsatisfactory

We carried out a Cognitive Dimensions (CD) assessment [7] to gauge the support of SUMLOW for exploratory UML design compared with conventional UML CASE tools. We plan to carry out an Attention Investment evaluation [4] to further assess its efficiency and effectiveness compared to traditional UML design tools. We summarise our CD usability results below:

- *Viscosity*. Some aspects of SUMLOW sketches require less effort to change e.g. redraw over top to resize, than conventional UML tools, while others require more effort e.g. movement is change mode/select/drag vs click-and-drag in most mouse-based tools.
- *Secondary notation*. The user has great freedom to sketch whatever secondary notation they desire in SUMLOW and to mix notational elements.
- *View support*. Multiple views are supported in SUMLOW, both sketched and formalised. Users can also mix notations within a view and sketch incomplete UML designs, providing greater modelling flexibility during early design work.
- *Closeness of mapping*. SUMLOW's sketched diagram elements must have a degree of similarity to computer-drawn UML elements in order to be recognised. This constrains the user to a degree and resolving mis-recognition can adversely impact on usability.
- *Terseness/diffuseness*. An almost infinite range of sketched shapes can be recognised 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*. Ambiguous sketches cause confusion for the 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.
- *Hidden dependencies*. The shape recognition algorithms employed by SUMLOW connect hand-sketched design elements to "assumed" formal UML elements within diagrams with limited user feedback.
- *Progressive evaluation*. The shape recognisers and user-demanded formalisation of SUMLOW design sketches support progressive evaluation within the tool.

In summary, our SUMLOW E-whiteboard UML design tool provides an efficient and effective sketching-based user interface on a large screen E-whiteboard. Both design sketch elements and text are constructed and manipulated using pen-based input. The freedom from heavily-enforced modelling constraints and flexible annotation facility in this environment encourages exploratory and collaborative UML-based software design. Disadvantages at present include recognition problems with some shape elements but particularly with the single-stroke text recognition algorithm. The learning curve for some gestures and current support for one-directional i.e. sketch to formalised UML model design are limitations we plan to address in future. Future work plans include the provision of distributed collaboration support utilising multiple E-whiteboards, support for bi-directional sketch to formal UML model and formal model to sketch transformation. The later will also support import of models from existing UML design tools and support sketch-based manipulation and annotation of these UML models for design review and re-engineering tasks. We aim to put our gesture-based sketching and E-whiteboard presentation support into a meta-CASE tool we are developing, making it much easier to "E-whiteboard" enable a very wide range of design tools in the future.

7. Summary

We have developed SUMLOW, a sketching-based UML design tool for E-whiteboard technology. Users collaboratively sketch UML software design elements, with sketched-based design elements recognised and hand-drawn shapes preserved. Users can flexibly annotate these sketches with their own sketched secondary notation and gesture-based manipulation of design sketches and single-stroke text recognition are supported. Users may mix UML design constructs in SUMLOW and may formalise their design sketches and have them exported to existing UML CASE tools as desired. Evaluation of SUMLOW indicates that this is a very promising approach to supporting exploratory, collaborative design.

8. References

1. Apperley et al (2002): Lightweight capture of presentations for review, In *Proceedings of IHM-HCI*, Lille, France, ACM Press
2. Apte, A. Vo, V. Kimura T. D. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In Proceedings of the 6th annual ACM symposium (1993) on User interface software and technology, ACM Press, pp. 121-128.
3. Berque, D., Johnson, D.K., Jovanovic, L. 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.
4. Blackwell, A.F. First steps in programming: A rationale for Attention Investment models. In Proceedings of the 2002 IEEE Symposia on Human-Centric Computing Languages and Environments, IEEE CS Press, pp. 2-10.
5. Damm, C. H. Hansen, K. M. Thomsen, M. Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard. In Proceedings of CHI 2000 on Human factors in computer systems: the future is here, ACM Press, pp. 518-525.
6. Fowler, M., Scott, K. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 2nd Edition, Addison-Wesley, August 1999.
7. Green, T.R.G. & Petre, M. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. Journal of Visual Languages and Computing 1996 (7), pp. 131-174.
8. Ideogramic, Pervasive UML, http://www.ideogramic.com/.
9. Iivari, J. Why are CASE tools not used? Communications of the ACM, vol. 39, no. 10, October 1996, 94-103.
10. Landay, J. A. SILK: sketching interfaces like krazy. In Proceedings of CHI'96 on Human factors in computer systems: common ground, ACM Press, pp. 518-525.
11. Lank, E., Thorley, J., Chen, S., Blostein, D. On-line recognition of UML diagrams, Proceedings of the Sixth International Conference on Document Analysis and Recognition, IEEE CS Press, 2001, pp.356-360.
12. Lin, J., Newman, M.W., Hong, J.I. and Landay, J. A. (2000): Denim: Finding a tighter fit between tools and practice for web design, Proceedings of CHI'2000, ACM Press, pp. 510-517.
13. Mimio® home page: from http://www.mimio.com (Last Visited Tuesday, March 26, 2003).
14. Myers, B.A. (1997): The Amulet Environment: New Models for Effective User Interface Software Development, *IEEE Transactions on Software Engineering*, vol. 23, no. 6, 347-365, June 1997.
15. Plimmer, B. Apperley, M. Computer-aided sketching to capture preliminary design. In Proceedings of the Third Australasian Conference (2002) on User interfaces, Australian Computer Society, Inc, pp. 9-12.
16. Quatrani, T. Visual Modeling with Rational Rose 2002 and UML, 3rd Edition, Addison-Wesley, October 2002.
17. Robbins, J. and Redmiles, D. Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML, Information and Software Technology, 2000.
18. Rubine, D. Specifying Gesture by Examples. In Proceedings of the 18th annual conference (1991) on Computer graphics and interactive techniques, ACM Press, pp. 329-337.
19. Smart Technologies Inc. (2002): *SMART Board*, www.smarttech.com.
20. UML™ Home Page, OMG (Object Management Group), available from http://www.uml.org/ (Last Updated Wednesday, March 19, 2003).
21. Voida, S., Mynatt, E.D., MacIntyre, B., Corso, G.M. 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.