

Chen, Q., Grundy, J.C., and Hosking, J.G. SUMLOW: Early Design-Stage Sketching of UML Diagrams on an E-whiteboard, *Software – Practice and Experience*, vol. 38 , no. 9, Wiley, July 2008, pp. 961-994

# **SUMLOW: Early Design-Stage Sketching of UML Diagrams on an E-whiteboard**

Qi Chen

Department of Computer Science

University of Auckland, Private Bag 92019, Auckland, New Zealand

John Grundy (Corresponding Author)

Department of Computer Science<sup>1</sup> and Department of Electrical and Electronic Engineering<sup>2</sup>

University of Auckland, Private Bag 92019, Auckland, New Zealand

john-g@cs. auckland.ac.nz

John Hosking

Department of Computer Science

University of Auckland, Private Bag 92019, Auckland, New Zealand

Phone +64 9 3737599

Fax +64 9 3737453

john@cs.auckland.ac.nz

## Summary

Most visual diagramming tools provide point-and-click construction of computer-drawn diagram elements using a conventional desktop computer and mouse. SUMLOW is a Unified Modelling Language (UML) diagramming tool that uses an E-whiteboard and sketching-based user interface to support collaborative software design. SUMLOW allows designers to sketch UML constructs, mixing different UML diagram elements, diagram annotations and hand-drawn text. A key novelty of the tool is the preservation of hand-drawn diagrams and support for manipulation of these sketches using pen-based actions. Sketched diagrams can be automatically "formalized" into computer-recognised and drawn UML diagrams and then exported to a 3rd party CASE tool for further extension and use. We describe the motivation for SUMLOW, illustrate use of the tool to sketch various UML diagram types, describe its key architecture abstractions and implementation approaches, and report on two evaluations of the toolset. We hope our experiences will be useful for others developing sketching-based design tools or those looking to leverage pen-based interfaces in software applications.

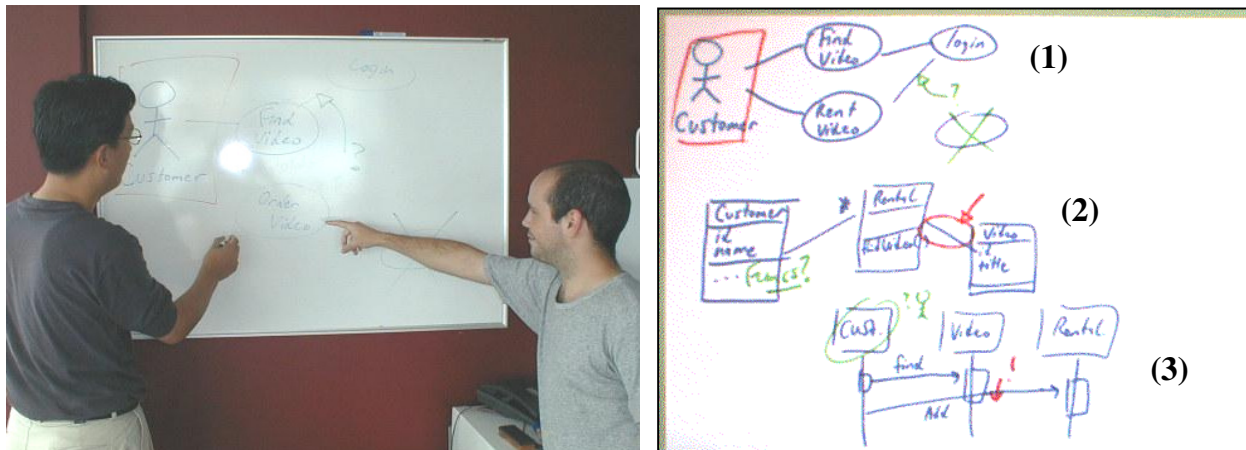
**Keywords:** sketch-based user interfaces, E-whiteboards, CASE tools, unified modelling language, hand-drawn visual language recognition

## Introduction

Software designers often use whiteboards when carrying out collaborative design work. These are used to collaboratively draw and discuss preliminary software design ideas, for example as whole or partial UML diagrams; to explore various possible architectural solutions; to assist in capturing high level code fragments; and to help organise design teams, schedule development activities and so on. An example of a conventional whiteboard being used for collaborative software design work is shown in Figure 1. In this example two software engineers are discussing aspects of the high-level design of a new software application. To the right are three example UML diagram sketches from this whiteboard that represent different aspects of the proposed software system: (1) a UML use case diagram, (2) a UML class diagram, and (3) a UML sequence diagram.

Some of the main advantages in using whiteboards for capturing such software designs include:

- *Immediacy*: whiteboards are very common and hence there is very little effort required to make a whiteboard available to designers, and it is very easy to create diagrams, capture text, and to delete or extend information;
- *Versatility*: a whiteboard can be used to capture diagrams of multiple, and even mixed, design notations. Additionally a variety of secondary notations - such as comments, arrows, highlighting, and colour – may be captured. Sketches on a whiteboard do not have to be precise nor complete in any formal manner, affording great flexibility;
- *Size*: a whiteboard is generally big enough to hold several design sketches and have relatively high “resolution”;
- *Collaboration*: a whiteboard allows multiple designers to discuss evolving designs, including taking turns at sketching and annotating designs on the whiteboard. Concurrent design work is even supported with multiple pens;



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

Disadvantages of whiteboards as a design tool include the lack of data persistency i.e. saving the content of the whiteboard for use later; an inability to export data to electronic design tools; the difficulty in easily making some changes to content e.g. repositioning or copying diagram parts; lack of distance (same or different time, different place) collaboration support; and ink dust. For these reasons much recent work has focussed on the development and use of large “electronic whiteboards” in a variety of design domains [34, 29, 4, 24].

In this paper we describe SUMLOW (Sketched UML On Whiteboard). SUMLOW provides an electronic whiteboard-based early design phase UML sketching and formalisation tool. SUMLOW allows UML diagrams to be sketched and recognised on an electronic whiteboard and then these designs to be exported to a conventional CASE tool. SUMLOW is an experimental tool we have used to explore the feasibility of using sketching-based techniques for early-phase requirements and design modelling on large-screen E-whiteboards. The key novelties of this proof-of-concept sketching-based design tool include:

- the preservation of hand-sketched design elements – unlike most other sketching-based UML design tools sketches are NOT converted to formalised, computer-drawn sketches immediately
- various pen-based manipulation facilities on sketches – users make use of various pen strokes to manipulate sketched diagrammatic elements
- ability to selectively formalise sketches to computer-drawn diagrams and support for exporting these to conventional CASE tools

In the following section we firstly review related research into conventional software design tools, sketching-based design tools, E-whiteboard technologies and sketching-based software design tools. We then describe the SUMLOW environment, architecture and design illustrating its use in a design scenario. Two evaluations of SUMLOW, a user survey and a Cognitive Dimensions-based analysis, are presented. We conclude with a summary of the contributions of our work and future research plans.

## Related Work

UML diagrams may be used at many times during the software lifecycle to aid design and documentation. During early design phases e.g. requirements engineering and high-level architecture analysis, diagrams are often more informal working sketches used to capture preliminary thinking. During detailed design and documentation phases, UML diagrams with more precise formal semantics are used to convey more details about software code. Model-driven engineering even uses such formal diagrams to generate code, possibly later modified in part by hand. Reverse engineering tools draw UML diagrams at low-level and high-levels of abstractions to show structure of target code.

Many software tools have been developed that support UML diagramming [11, 37], most of them to support modelling of software designs. Almost all of these tools adopt conventional mouse and keyboard input and standard monitor display of information [30, 31]. Take-up of such tools within organisations has been mixed at best [22, 5, 26, 15]. Empirical studies show that designers find conventional CASE tools to be overly restrictive, during early design phases in particular, often leading to poor utilisation of the tools [22, 24, 9]. A number of HCI studies have shown that developers will prefer sketching designs by hand rather than using a keyboard and mouse, especially in the early stages of software design [9, 23, 29]. Goel [12] and Black [5] have shown that designers reject tools employing conventional mouse-driven click-and-edit operations during early design-phase work. Diagram editing constraints can also be very distracting and off-putting to users, especially during creative design work [7, 22].

Many approaches have been adopted to support collaborative work in conventional CASE tools and other design environments [16, 17, 33]. Most of these approaches have focused on same time, different place (distributed synchronous) or different time, different place (distributed asynchronous) collaboration. In general these tools have also not had very successful uptake to date. Several studies have shown that use and sharing of hand-drawn sketches can better-support collaboration during early-phase design [39, 13, 9]. In particular, same time, same place collaborative early design phase sketching approaches have proved to be successful in other domains [9, 39].

A number of design tools have been developed using a pen/sketch-based approach, in order to overcome some of the problems of mouse-based design environments. One of the earliest examples, SILK [23], provides a tool for software designers to sketch a user interface design using an electronic pad and stylus. SILK attempts to recognize the sketched user interface elements as soon as they are drawn and to formalise them. This approach is not intrusive, however, and users are only made aware of the results when they choose to experiment with the design elements. The user interface element recognition algorithm uses Rubine's single stroke gesture recognition algorithm [32]. SILK can transform the sketches of user interface elements into standard Motif widgets and graphical objects when the designer has completed their sketching work. The designer can switch the sketch into run mode to test the interface. Annotation and editing are also supported by SILK to allow designs to be marked up to facilitate discussion and for sketched designs to be modified easily. However, SILK only recognizes a few ways of drawing each widget and does not support specification of widget behaviour. Other related user interface work includes web-based user interface design in Denim [25] and prototype gesture-based document manipulation tools [1, 28].

Electronic whiteboards have become increasingly popular to support a number of design-related activities. These include facilitation of meetings with collaborative document display and annotation [38]; computing education [4];

Power-Point™ presentation control and annotation [1]; and (early phase) software design [24]. The previously listed main advantages of conventional whiteboards are partially replicated with E-whiteboard applications, with additional advantages of data capture and display, distributed work support and control by gesture-based interfaces [1, 4, 34].

FreeForm [29] is a Visual Basic (VB) add-In for the design of VB user interface forms. FreeForm uses the same metaphor as SILK but utilises an electronic whiteboard to support sketching-based UI design. Shape and character recognition both use modified forms of Rubine's [32] algorithm. Plimmer discovered via experiments with end users that retention of sketches while doing user testing of forms improved quality of testing over conversion to VB forms and hence the quality of the software design.

Knight is a widely-known UML design tool that uses gestures on an electronic whiteboard with pen input [9]. A number of similar tools have been developed for UML-based software design with sketching-based input of diagram elements [24, 21]. Some preliminary work has also been done on recognising UML diagram shapes from glyphs [24]. Most of these tools support UML diagramming by stroke recognition and then immediate conversion of sketched elements into computer-drawn, formalised UML diagram elements. Knight uses two methods to support input and recognition - compound gestures combine gestures that are either close in time or space to form one drawing element. Eager recognition uses Rubine's algorithm to try and classify shapes while they are being input by the user. Text input is supported by keyboard, on-screen Virtual Keyboard, or stylus-based gestures. Most existing UML sketching tools currently adopt immediate recognition and formalisation of UML diagram elements. Some allow incomplete elements to be recognized at a later time. However, there is usually no association between a sketched element and a formalised element, the sketched content usually being lost once recognition has been done.

Multi-stroke recognition systems often use a variety of extensions to Rubine's single-stroke algorithm to try and improve their accuracy and robustness. Several of these have been developed and applied to recognising geometric shapes drawn with multiple strokes [19, 2], which allows for richer sketched element recognition. From this work a number of sketch recognition engines have been developed which provide support for training multi-stroke recognisers in a number of domains [20, 36]. Many of these approaches have demonstrated high degrees of accuracy but have tended to be applied to a variety of generic geometric shape recognition applications rather than UML-style design modelling tools. However examples of multi-stroke sketch recognition tools for some UML diagram types have been developed, including activity diagrams [10] and class diagrams [18]. These tend to focus on supporting a small range of quite constrained UML diagram elements and usually enforce normal UML diagram editing constraints on these sketched diagrams. Some of these multi-stroke recognition systems provide on-the-fly formalisation of sketches to computer-drawn content but most focus on preserving sketched content and formalising the diagram content on user request. Most aim to support pen-based drawing and design as an alternative to non-pen based tools rather than early-phase design sketching prior to conventional design tool usage.

## **Our Approach**

Given these demonstrated benefits of sketching-based interfaces for early phase design tasks, we wanted to develop a UML-supporting early phase design tool that leveraged these advantages while overcoming some of the disadvantages of

previous work in this area. The primary motivation in developing our SUMLOW tool was to retain as far as practical the hand-drawn sketch “look and feel” of real whiteboards with UML sketches, while providing better editing capability and the ability to recognise and convert the sketches to more formal diagrams. This approach is very different from most other sketching-based design tools for software design applications where sketched elements are typically converted into computer-drawn formalised elements after each manipulation of the diagram. In particular we were influenced by Plimmer’s observations in her Freeform work for user interface layout design that retaining a sketch form encourages more experimentation with design [29]. We also wanted to overcome some of the problems identified in work by Brooks and Scott [7] relating to the interference of UML diagramming constraints on creative design in conventional UML CASE tools. We relaxed many UML graphical notational constraints during sketching to allow designers freedom to sketch partially-compliant diagrams, with the ability to apply constraint checking during user-requested diagram formalisation. Thus conventional UML diagram editing constraints are relaxed during sketch-based design and applied when the users decide to formalise their sketches.

We wanted an environment where same time, same place face-to-face collaboration between UML designers would be supported by the sketching-based interface and large-screen presentation of designs as afforded by an electronic whiteboard. Our aim in this research was not to support same time, different place collaboration with multiple E-whiteboards, though that is an area we plan to explore in future research. We wanted to explore same time, same place E-whiteboard early phase design collaboration as is currently done with conventional whiteboards.

These motivations and goals were elaborated into a set of requirements, which were also informed by a survey of experienced UML designers who expressed their opinions on desirable system features. A summary of SUMLOW’s requirements is as follows

- A pen-based electronic whiteboard system that supports sketching of a core set of UML diagrams while retaining the sketch “look and feel”
- Progressive recognition and formalisation of diagram elements but with minimal interruption to the user while sketching and with good performance and accuracy of the capture and recognition algorithms used
- Early design phase work the focus, with UML diagram constraints not interfering with the sketching process.
- Functionality to edit sketched diagrams without destroying the sketch look and feel
- Ability to use a wide variety of secondary notation, such as colour, annotations, etc
- Support for persistency of sketches and recognised UML diagrams and constructs, and the ability to export to a conventional UML tool
- Same time, same place face-to-face design collaboration supported via a large screen electronic whiteboard and either multiple or shared sketching pens

Our approach to meeting these requirements is an electronic-whiteboard based sketching tool that recognises UML constructs as they are drawn but preserves the hand-drawn look-and-feel interface throughout the design phase. This differs significantly from other existing UML sketching approaches like Knight [9, 21, 24]. In SUMLOW the look and feel of the hand-drawn constructs are always retained while still allowing sketched elements to be moved, resized, 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, colour, arbitrary sketches, etc.

The need to support early design phase work meant that, unlike in almost all UML-based CASE tools, UML diagramming constraints are not enforced until the designers explicitly request this. This was due to the influence of the empirical evaluations of conventional CASE tools by Scott and Brookes. They have shown that conventional UML CASE tools poorly support early design phase work due to over-enforcement of diagram editing constraints [7]. SUMLOW is unusual in that elements from different UML diagram types can be mixed together during early phase design in ways that may violate a UML diagram's syntax and semantic constraints. We supported this based on user feedback where experienced designers wanted to be able to mix UML constructs without constraint during conceptual design work. Diagrams or parts of diagrams can be progressively formalised as desired then checked for feedback on syntactic and semantic constraints, and exported to a standard UML design tool. Figure 2 shows SUMLOW, in use. We have used a Large Image Display Surface (LIDS) electronic whiteboard [1] with a backlit display (this eliminates shadowing), and combined with a Mimio [27] ultrasonic system for pen location. However the approach is applicable to other pen-based electronic whiteboard systems e.g. Smartboards.

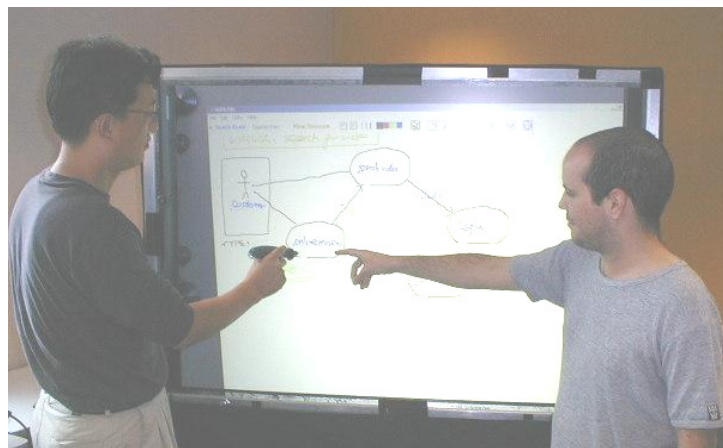
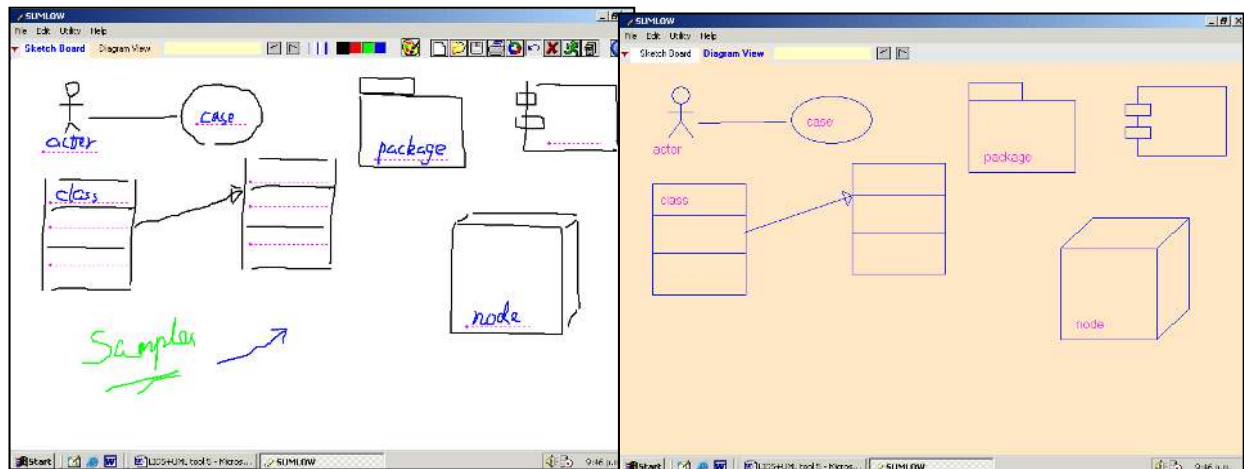


Figure 2. Our SUMLOW design tool in use, from [8] © 2003 IEEE.



**Figure 3. SUMLOW in use showing various recognised UML constructs in sketch view (left), and formalised elements in diagram view (right) - from [8] © 2003 IEEE.**

Figure 3 (left) shows SUMLOW's sketching view. In this example several UML constructs, together with annotations on the diagram, have been sketched. UML elements can be moved and copied, for example the right hand class construct was copied from the left one. Text entry areas – dotted pink lines – have been added automatically to constructs that have been recognised. These text areas indicate to the user where to add names, attribute information, etc. This example shows several UML elements mixed in the same diagram.

Figure 3 (right) shows SUMLOW's formalised design view. This displays recognised UML shapes in a formalised, computer-drawn form. Note in this view annotations are omitted as they are not formally recognised constructs. When exporting formalised UML diagrams to a CASE tool, diagram elements belonging to particular UML diagram types are filtered and included in multiple UML diagrams exported by the tool. 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. Designers may exchange the sketching pen at any time to support collaborative design, or each designer may have their own pen.

## **An Example Design Scenario**

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.

### *Design Scenario Overview*

Figure 4 shows a few example UML diagrams for the video rental application. The use case diagram (1) describes some of the basic user interaction with the system; the class diagram (2) shows some of the main organisations of data and operations; and the sequence diagram (3) one of the flows of operation invocations between objects. John and Michael want to use SUMLOW to collaboratively develop these kinds of design diagrams via their E-whiteboard, sketching-based user interface, rather than a conventional UML tool.



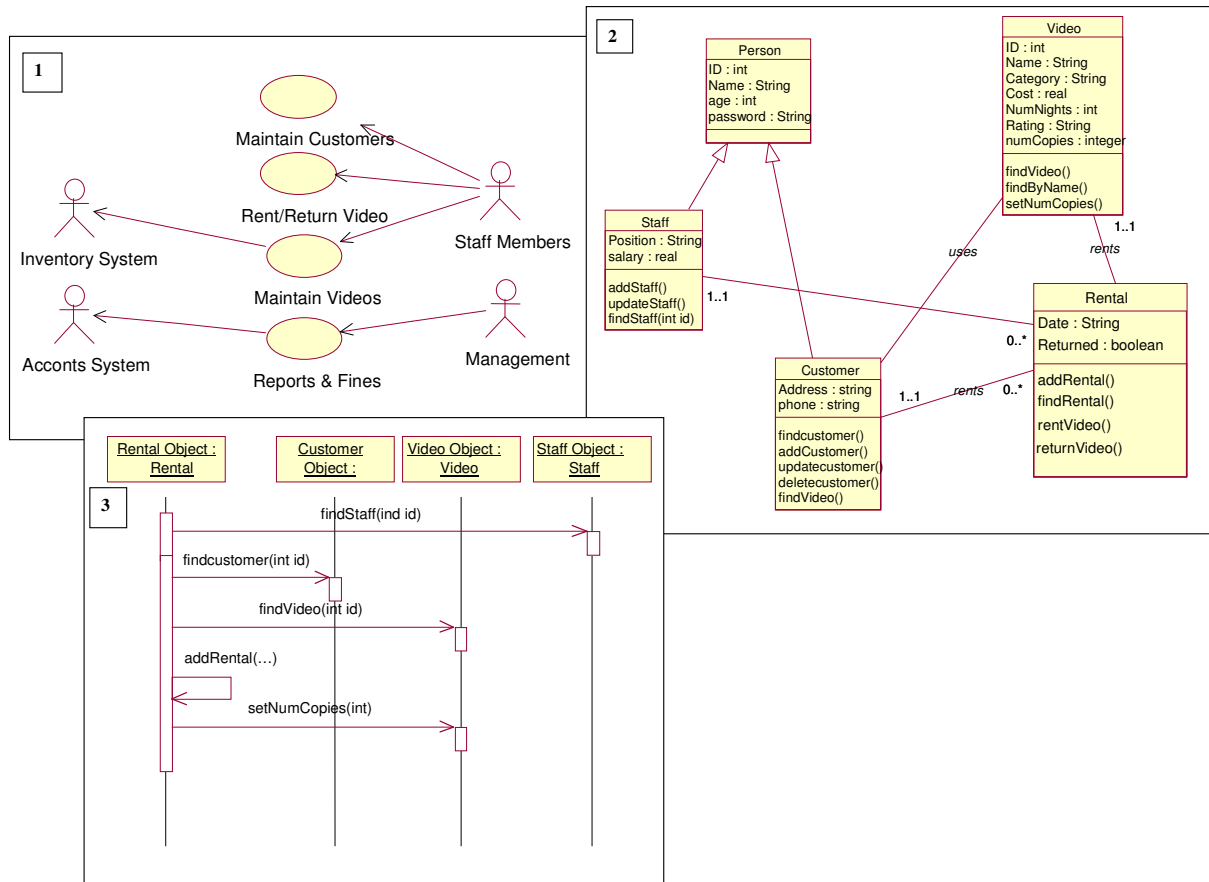
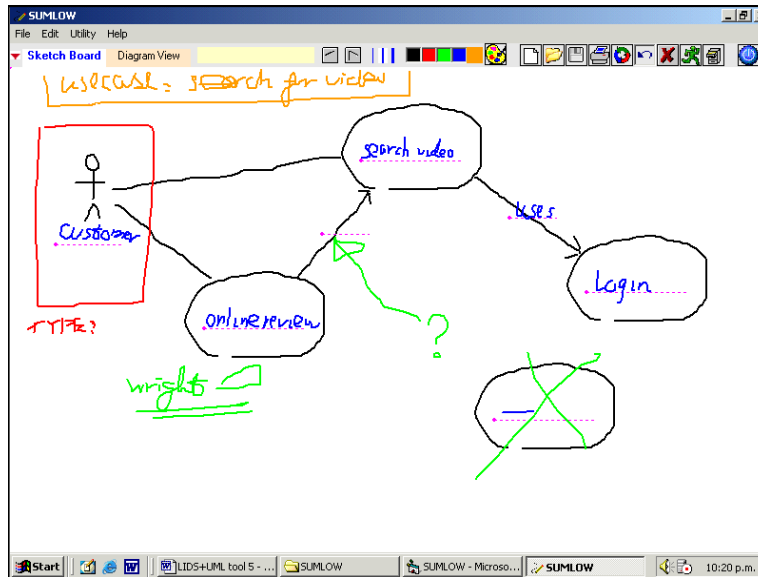


Figure 4. UML design examples for the video store application .

### Use Case Sketching

John begins by sketching out the main “use cases” (groups of user-system interaction requirements) using UML use case diagram elements: actor shapes connected by 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 annotations. Shapes are manipulated with pen gestures to indicate movement, and deletion. SUMLOW carries out a simple redrawing algorithm to redraw sketched connector lines.



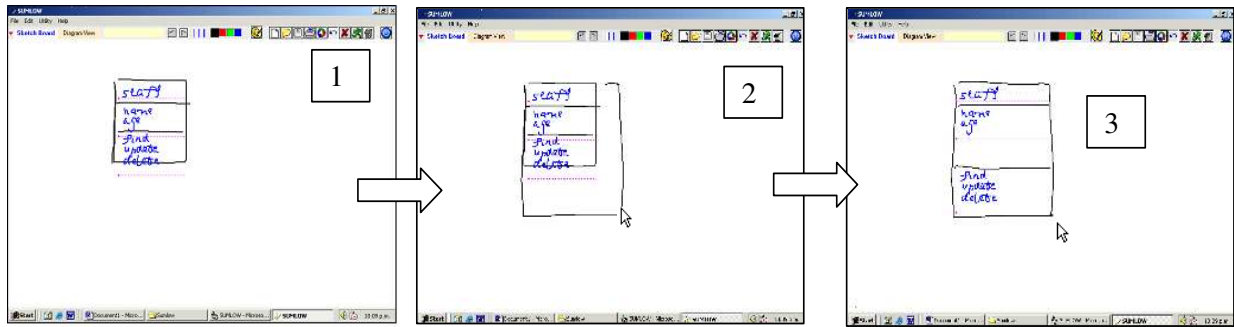
**Figure 5. Use case model sketch in SUMLOW, from [8] © 2003 IEEE.**

Figure 5 shows the resulting use case diagram in SUMLOW. An actor (“Customer”) has been drawn along with four use cases (search video, onlinereview, login and an untitled use case). Connections between some elements have been drawn, along with some custom annotations. John and Michael have each added 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. These can be preserved to document the “design rationale” or removed later if desired. They are not standard UML diagram elements and are omitted when the diagram is formalised.

John and Michael may share the construction of the use case diagram by either swapping a single mimio pen or by turn-taking with a pen each. SUMLOW does not currently distinguish between different mimio pen input and so treats input from each user as if it is from a single user. Thus social turn-taking protocols are used to resolve syntactic conflicts e.g. trying to draw in the same place or one designer trying to delete an item while another modifies it. Semantic conflicts e.g. same-named use cases or incomplete models, are detected and resolved later when a diagram is formalised (see discussion below).

### *Class Diagramming*

After John has sketched out these use cases, Michael takes over to sketch out initial classes and relationships. Class shapes are quite complex. Our multi-stroke recognition algorithm recognises these and adds three text entry areas to the shape, one for each kind of text item the user can draw. Figure 6 (1) shows the result of drawing the first class, “Staff”. As Michael writes names for attributes and operations on the respective text entry lines the insertion point moves to accommodate additional entries.



**Figure 6. Sketching class icons in SUMLOW and an example of a resize sketching-based manipulation.**

In this example, it can be seen that Michael has drawn his class too small for the additional textual data. In a conventional tool either (1) the user will resize the icon by direct manipulation of “handles” on its border or (2) the tool will automatically resize the icon. We experimented with both of these options for SUMLOW but found neither worked particularly well – automatic resizing is difficult as the size of text to be entered inside the class icon is dependent on the user’s needs. SUMLOW automatically resizes class, use case, object, component and machine icons to fit a text entry annotation, but these sometimes do not suit the user’s need. Resize via handles we found to be contrary to the “electronic whiteboard” metaphor of SUMLOW. As an alternative we developed a *replace* paradigm whereby the bounds of a construct are redrawn to indicate the size of the replacement, as in Figure 6 (2). Any existing sub-elements of the sketch are transferred across to the new shape, as shown in Figure 6 (3). We found this to be a much quicker and more natural action for users to perform and our evaluations of SUMLOW have found this to be an intuitive operation for users. Note that SUMLOW repositions the class name, attributes list and methods list itself. This demonstrates the advantage of an E-whiteboard over a conventional whiteboard interface, where the user would have to redraw the entire class by hand for a “resize” operation.

A sketching-based diagram manipulation paradigm has been applied where feasible to other operations like move, delete, bring forward/send backward and so on. For example, to move a SUMLOW sketched element the user draws a border around it, selecting the enclosed sketched element(s). The user then drags the selected sketch(es) to a new position, having them redrawn by SUMLOW. A selected sketch may instead be deleted by a cross gesture through it, we implemented in response to user feedback on the previous modal deletion tool. A selected sketch is moved in front of another overlapping one by upwards gesture beside it, and downwards moves it behind. De-selection of a sketch is by tap beside it or selection of another.

Sometimes, as in conventional UML diagramming tools, diagram elements may overlap or otherwise interfere with one another. SUMLOW uses the whiteboard metaphor to solve these – as on a conventional whiteboard, UML diagram elements may overlap and they are displayed in this manner. Shape manipulation gestures on overlapping parts e.g. a resize are disambiguated and applied to either whichever shape was wholly selected or to the front-most shape or part of a shape. The user may also use this technique to force SUMLOW to re-recognise a sketched shape as a different UML diagram element. SUMLOW currently uses no automatic layout algorithm as these have been shown to significantly interfere with early design phase work [7]. Users must manipulate diagram elements explicitly to affect resizing and moving.

One difference between SUMLOW and conventional UML tools is when a modelling icon e.g. class shape, is deleted. In conventional tools, relationship icons disappear if either associated end is deleted. SUMLOW users often sketch another element to replace the one deleted and prefer the relationship to be kept instead of having to draw another one. The user may manually delete a “dangling” relationship if necessary with a single cross gesture over it with the pen. As the kind of relationship between icons is dependent on the connected icon types themselves, recognition of relationship type is deferred until both source and target icons have been recognised. If one of these is deleted, the relationship type is made “unknown” until another source or target has been drawn and recognised by SUMLOW.

A more complete UML class diagram sketch is shown in Figure 7, with several classes, associations and generalizations. Here, Michael has named classes and added attributes and operations to three of them. An extra use case sketch is added at top left (boxed off). During design discussions 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.

John and Michael may revisit their use case diagram(s) during or after class diagram design and may use constructs from use case diagrams in the class diagrams, as shown in Figure 7. Conventional UML CASE tools disallow this mixed use of diagram elements, enforcing a rigid definition of diagram types and symbols. SUMLOW in contrast aims to provide more flexibility to the designers to mix and match notational symbols as they wish during sketching design.

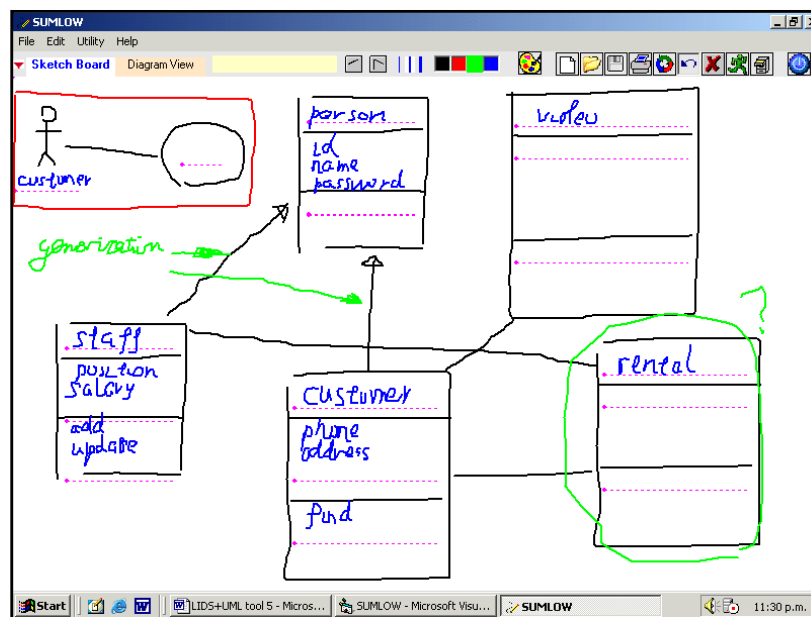
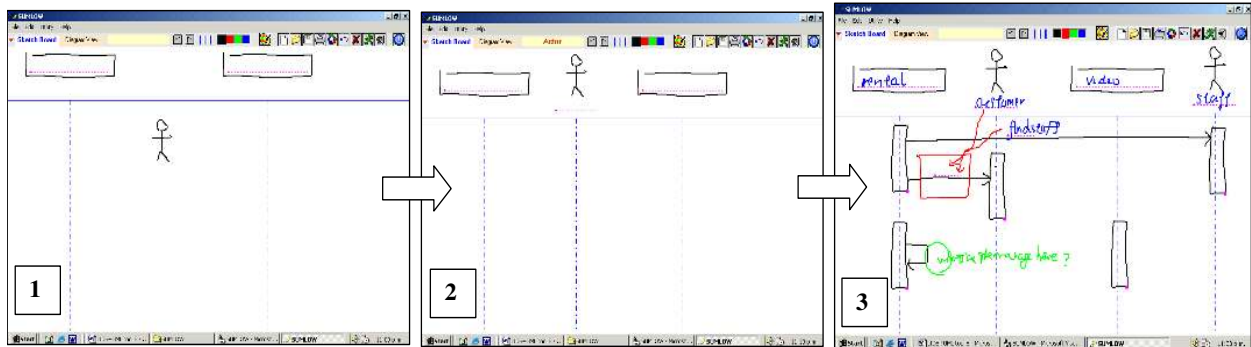


Figure 7. UML class diagram sketch in SUMLOW, from [8] © 2003 IEEE.

### UML Sequence Diagrams

After discussing and modifying the initial class diagram sketch, John and Michael focus on a complex message flow in the video system design, sketching a sequence diagram to capture and discuss this dynamic system behaviour. Figure 8 shows this, with objects, timelines, and operation activation and invocation. Here John and Michael have also used Actor

shapes instead of object rectangles for two objects, customer and staff. This violates UML diagramming conventions, but is useful for assisting John and Michael to discuss their early phase design sketch.



**Figure 8. UML sequence diagram sketching.**

As sequence diagrams are complex and require space, other diagram types cannot be mixed with them. Initiation of a sequence diagram sketch is done by drawing a horizontal line across the top of the sketch board. At that point, other 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.

For example, in Figure 8 (a), John has drawn a UML Actor icon below the horizontal bar separating objects from method invocations. To preserve some semblance of UML sequence diagram semantics and potential to formalise the diagram, SUMLOW moves the Actor icon to the object list Figure 8 (b), repositioning the objects and any connected method invocation lines. After adding more objects, method invocation lines and invocations, the resulting sequence diagram is shown in Figure 8 (c). Note the use of additional annotations on the sketch which are secondary notation and not formalisable.

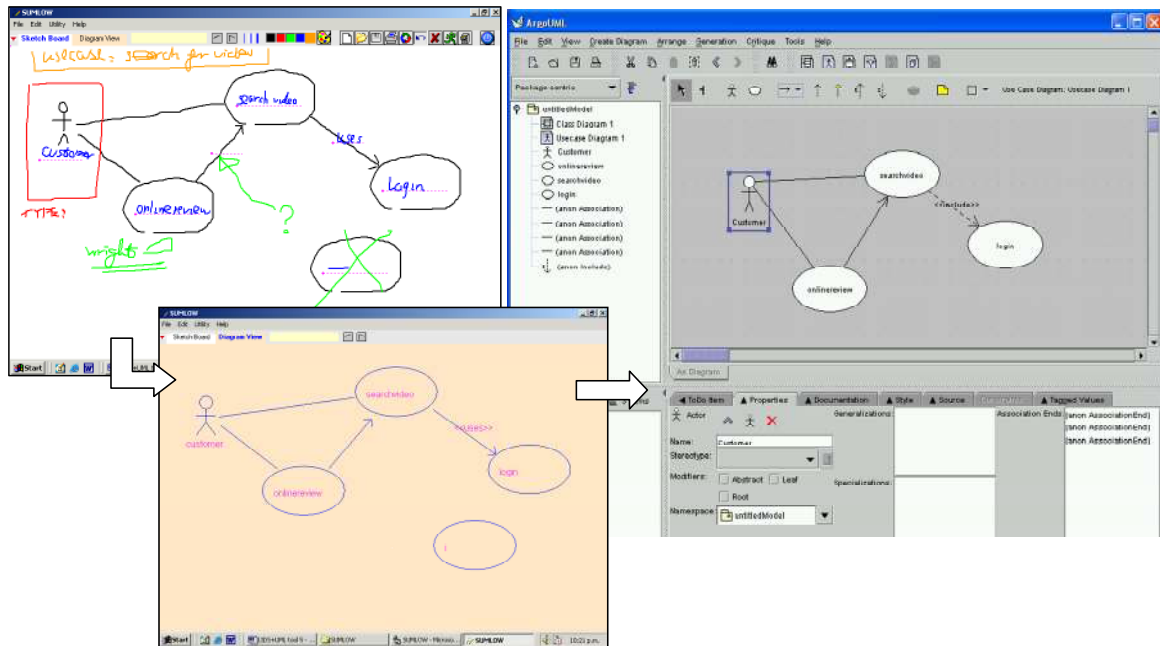
### *Recognition and Formalisation of UML Symbols*

As John and Michael design using the sketch board shapes are recognised as UML element types in a background process. After a sketched element has been recognized its UML element type is used to assist the recognition of sub-elements (inside or on the element’s border) and relationships between icons. Typically a relationship in a UML diagram can only exist between two recognised UML diagram elements. However during sketching SUMLOW allows users to draw “dangling” relationships or to delete the source or target icon for a relationship leaving the relationship intact. While such dangling relationships may exist in a sketch they can not be recognised as a UML relationship type by SUMLOW if they have no source or target icon or are not close enough to two elements to be regarded as “connecting” them. If a source or target icon is deleted, the relationship becomes unrecognised until another is drawn in its place, when the shape recognition process attempts to re-categorise its UML relationship type.

SUMLOW provides a “UML design view” which renders the recognised UML elements in a conventional computer-drawn UML diagramming notation format. A design view is populated by an underlying formalised UML data structure



UML CASE tool, as these tools typically do not support mixing of UML diagram notations. Figure 10 shows an example of formalised SUMLOW diagrams exported to the Argo/UML CASE tool. The use case diagram shown here has been converted into the XMI format by SUMLOW, imported into ArgoUML and the ArgoUML model data structures populated. We had to layout the use case diagram by hand in ArgoUML as shown here as locality of icons is not supported directly in the XMI interchange file.



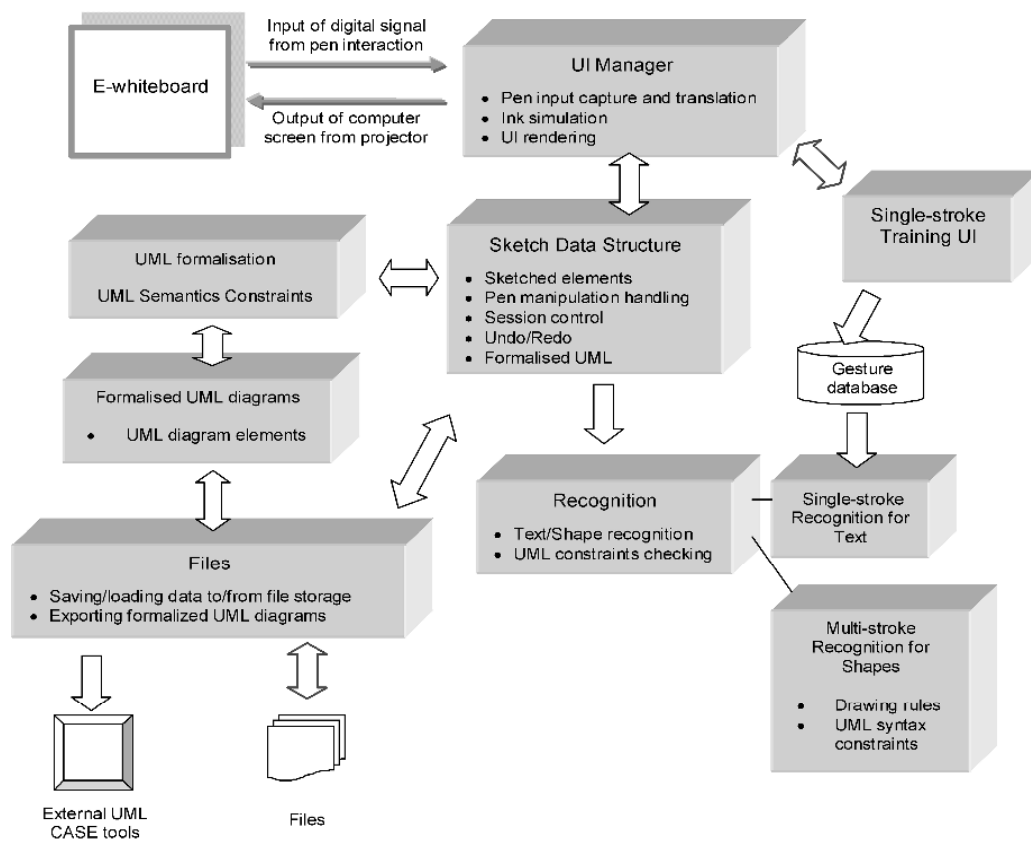
**Figure 10. Example of exported UML diagrams from SUMLOW to the Argo/UML CASE tool**

## Architecture and Design

We were faced with a number of design and implementation challenges when developing the SUMLOW environment. These included:

- Capturing pen-based input on a sketching canvas rather than mouse click-and-move and keyboard-based data input
- Recognising UML modelling elements from multiple pen strokes
- Recognising characters from pen strokes (for labels, names etc in diagrams)
- Supporting manipulation of UML diagram elements by pen-based gestures rather than conventional point-and-click direct manipulation techniques
- Preserving the sketched look-and-feel of diagram elements under gesture-based manipulations like move and resize
- Storing complex UML sketched diagrams, formalised diagrams and their underlying design models in appropriate data structures
- Generating XMI interchange format files to export SUMLOW formalised designs to conventional UML CASE tools

Figure 11 shows key architectural components of SUMLOW. It consists of: 1) an E-whiteboard and its input/output device; 2) software components (represented as grey cubes), 3) data storage (XML files and a gesture database); and 4) internal/external communications (represented as arrowed lines). A set of data structures represent the sketch content of a diagram as a set of grouped line points (single strokes) and groups of single sketches (multiple stroke sketches). A second data structure represents formalised UML content with recognised UML elements linked to their single- or multi-stroke sketch input. These data structures may be saved and loaded to SUMLOW files and the UML elements exported to an XMI tool inter-change format. User input manipulates the sketch data structures with the recognition system driven by events generated by changes to the sketch data structure. Sketched elements are annotated with a recognised element type which is used to inform further recognition of related sub-elements and relationships, and to assist user interaction e.g. by the tool proactively adding text entry annotations to the sketch. A single-stroke text recogniser is currently used to identify text and stores the text value with sketched elements. On user request the sketched content is converted into the formalised UML data structure. The following sections explain the design of each of these key parts of SUMLOW’s architecture.



**Figure 11. The SUMLOW application structure.**



## Data Structures

SUMLOW uses a set of data structures to maintain the sketched diagram elements, formalised design diagram elements and their underlying UML model constructs, a gesture database for text recognition, and undo/redo and session control for users. The meta-model of the key parts of this formalised UML representation is shown in Figure 12. This data structure is saved and loaded to an XML file representation by SUMLOW. It can also be converted into an XMI representation for export to other XMI-compliant UML tools like ArgoUML and Rational Rose.

Sketches are made up of one or more “Stroke” objects, with each stroke represented as a set of connected points. Mouse movement events from the user interface are used to create each Stroke and these strokes are grouped into a sketch element by the shape and text recognisers. A sketch may have been recognised as a UML construct – i.e. a set of strokes making up a UML element, or have been recognised as text i.e. a set of strokes making up a line of text. If the shape recogniser identifies a set of strokes as relating to a UML element then it groups these strokes into a single sketch element and indicates the UML element type that the sketch corresponds to. Text is recognised by a separate, single-stroke text recogniser. The text recogniser groups a set of strokes, each stroke corresponding to a single character, into a line of text and the textual string corresponding to the sketched text input. Formalised UML design diagrams are represented in a conventional object-oriented manner, but with a relationship to the sketch element they are derived from. A model-view approach is used to associate sketched elements and formalised design diagram elements to UML model elements. This supports multiple views of a model element in different SUMLOW sketch diagrams and formalised design diagrams. We developed a data structure loosely based on the UML meta-model to hold our formalised UML diagram information in SUMLOW.

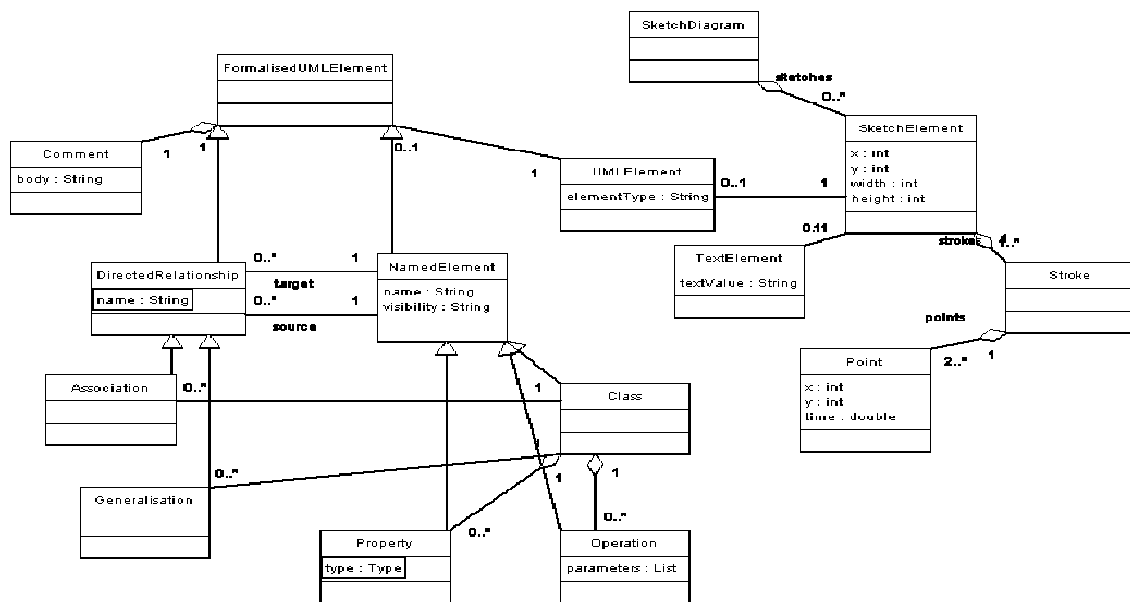
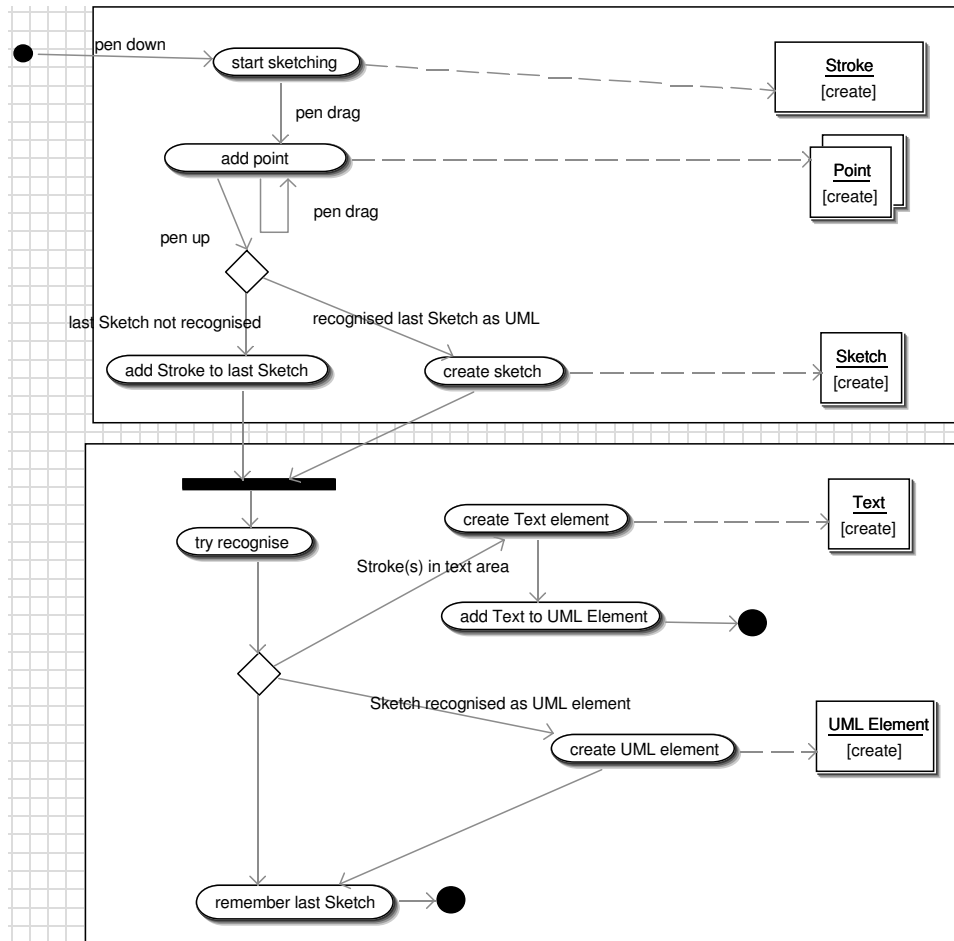


Figure 12. Outline of SUMLOW sketch and formalised UML element data structures.

### *Recognition Process*

Figure 13 shows an outline of the process used when drawing Strokes, recognising a set of strokes as a UMLElement or Text, and grouping Strokes into Sketch elements. Initially the user begins sketching by repeatedly putting their pen down, dragging it, and releasing it. Each of these actions creates a Stroke which records the pen movements as a set of Points. At each pen up, SUMLOW either creates a new Sketch element with the new Stroke (if the last Sketch was recognised) or attempts to recognise the new Stroke when added to the previous Sketch (if certain timing and proximity heuristics are met i.e. the new Stroke occurs very recently after the previous was drawn and some Point within it overlaps or is close to the previous Sketch's Strokes). SUMLOW then tries to recognise the composite Sketch element as text – if it has been drawn above an annotation (illustrated previously) - or as a UML Element (using the multi-stroke recognition process described below). The text recogniser assumes one Stroke per character (described in detail below). The UML element recogniser uses a multi-stroke recogniser (also described in detail below). If text is recognised, a Text object is created and added to the UML Element the text annotation belongs to. If a UML element is recognised, a UMLElement object is created and its type property set appropriately.

As shape and text recognition – and human sketching - are inherently imperfect, the SUMLOW user can always: force a retry of the shape or text recognition process; manually ungroup Strokes from a Sketch and regroup them into one or more new Sketches; or force SUMLOW to recognise a Sketch as a particular UML Element or Text value.



**Figure 13. An outline of the SUMLOW shape and text recognition process.**

### *Multi-stroke Recognition for Shapes*

Due to the complexity of the UML diagrams and our initial experimentation with simple recognition algorithms we found that no single existing approach was a complete solution to recognizing sketched UML notations. UML element parts can, however, be categorized into iconic and textual components. Our approach is to have users draw elements in two steps: first a shape representing the UML element icon is drawn and recognised. Text entry fields are then automatically added by SUMLOW as annotations to the sketched shape. The user can sketch text above the text annotations within the shape boundary. This allows us to use an effective multi-stroke recognition algorithm to recognise diagram element shapes and a single-stroke algorithm for text. It also allows us to associate the right textual value entered with the right underlying UML element text property.

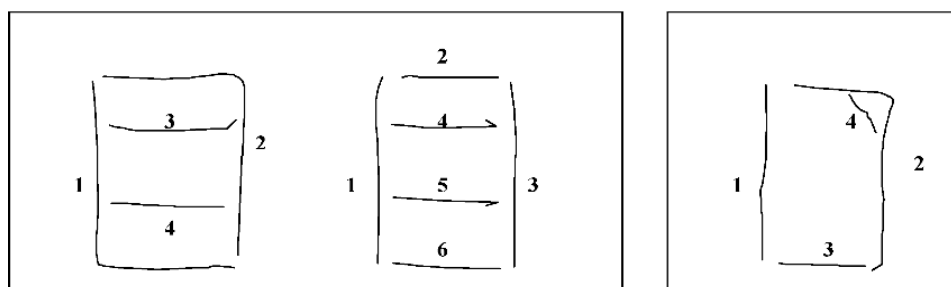
We chose to use a multi-stroke recognition algorithm that we extended from the work of Apte et al. [3] for UML diagram icon recognition. This has the advantages that:

- more complex shapes including shapes with sub-shape annotations can be handled than with a single stroke algorithm

- unlike tools that use a single stroke approach the appearance of the sketch corresponds closely to the UML notational shape as it can be made up of multiple strokes
- multiple stroke drawing of UML shapes supports a more natural style of interaction for users when composing drawings from multiple, simple parts
- Apte's algorithm requires no training stage, unlike most single-stroke and many multiple-stroke approaches that require considerable per-gesture and per-user based training to gain sufficient recognition accuracy

The main disadvantage of this approach is that the user must draw a UML icon in one or several pre-defined ways, else SUMLOW will fail to recognise it as the correct UML element type. While SUMLOW does support drawing of the same UML shape in several ways, not all possible ways they could be drawn by users are recognised. Thus users need to familiarise themselves with SUMLOW's supported ordering of UML icon drawing in order for shape recognition to be effective. Some users we have surveyed have found this to be less than ideal.

Apte's original algorithm supports recognition of simple, single geometric shapes like squares, rectangles, ovals and so on. It provides a filtering technique that takes multiple stroke input and categorises each of these strokes into a simple geometric shape. However, this is not sufficient to recognize the complex UML icons which are mainly combinations of geometric shapes. This is especially so when a user sketches the shapes in multiple strokes. Our extensions to Apte's algorithm provide a second level of recognition filtering, recognising strokes within geometric shapes, and a sizing comparison, that checks if adjacent shapes are correctly proportioned, and that now allows combinations of fundamental shapes to be recognised as quite complex UML notational symbols. For example, in Figure 14, the left hand shapes are Class icons sketched in two different orders and stroke numbers, both recognised by SUMLOW as a UML class icon, while the right hand shape is a sketched UML Note icon. The numeric figures show the ordering and the number of strokes. If we feed the above three sketches into the filters contained in the original Apte Multi-stroke Recognition Algorithm, the output will be the same – a rectangle shape (the algorithm ignores any strokes within the boundary of the sketch).

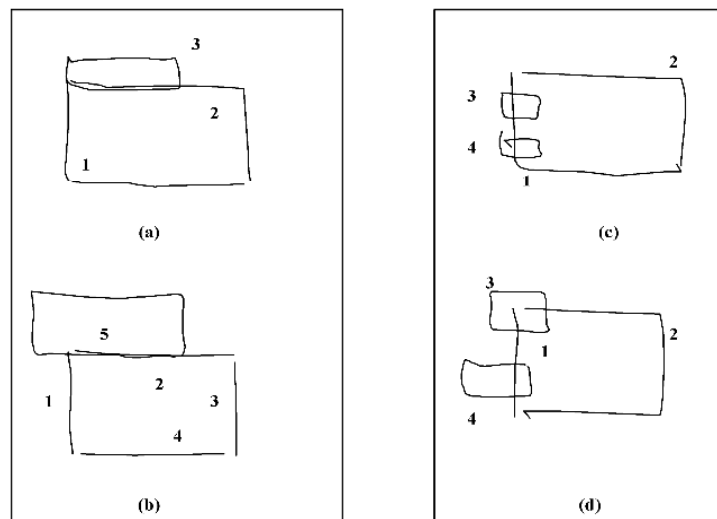


**Figure 14. Recognising UML icons with our extensions to Apte's algorithm.**

A second-level filtering method needs 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). We developed a “drawing rule” technique which contains ‘rules’ of how an UML icon has to be assembled by the user from its sketched

parts. Firstly, the drawing rules define 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 or a single stroke, and then the two inner lines in either order. 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.

The drawing rules also define the relative position of strokes and the size of the sketch for a UML symbol. For example, in Figure 15 the user has attempted to sketch the Package icons (a) and (b), and the Component icons (c) and (d). The sketches in (a) and (b) both follow the drawing rule order (the head square is drawn last) but the sketch in (b) does not look like the Package icon as the head is in a wrong position and too large. The sketch (a) is recognized as a Package icon, but (b) is not due to its second rectangle shape being wrongly positioned and sized. Similarly the Component icon (c) is recognised but (d) is not.

















**Figure 15. Recognized (a, c) and unrecognized (b, d) sketches in SUMLOW.**

The first-level filtering algorithm of Apte returns the most probable 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 1. This process repeats for several iterations until a UML notation type or a ‘Not Recognized’ result is returned. The abstraction of UML notations into geometric shapes for recognition by drawing order is unique to our application. Table 1 summarizes the categorization of UML diagram element notations we have implemented in SUMLOW to date. In SUMLOW we represent each of these drawing rules as a list of fundamental geometric shape types each with a relative position and size, and any associated text properties and text annotation positioning information. Currently these drawing rules are hard-coded in the tool and don’t allow users to modify or extend them directly.

Our approach to using a fixed drawing ordering of the simple shapes that make up a more complex UML symbol has advantages and disadvantages. The key advantage is that recognition rates for UML notational symbols are very high (see the Evaluation section) and the text annotations on drawn symbols can be accurately sized and positioned by SUMLOW

for text entry. However this is at the expense of SUMLOW only being able to recognise pre-determined symbols that have been drawn with pre-ordered combinations of simple geometric shapes to make up the supported UML diagram elements. The main impact on the user is the need when sketching UML symbols to order appropriately the way in which the simple shapes making up a symbol are put together. Only some possible orderings of the possible simple geometric shapes that could be used to draw the symbol are recognised by the tool. We have put considerable effort into selecting orderings that were most natural to users for construction of each element type based on observations of users sketching UML designs. Our usability studies show that recognition is very efficient with all shapes recognised within 2 ms and with a very high degree of accuracy. An extension to SUMLOW would be to allow users to re-order the geometric shapes for a UML symbol and change their relative size/positioning via a reconfiguration interface as used in Plimmer's user interface design [29].

	Icon	Shape(s)	Text
<b>Major Notations</b>			
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
<b>Relationship Notations</b>			
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 1. UML diagram notation element categorization in SUMLOW.**

After a UML icon symbol is recognized it is then checked against basic UML syntactic and semantic constraints. 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 having been drawn or is floating (ie not close enough to two major notations).

Such constraints are currently hard-coded in the multi-stroke shape recogniser as a third-level filtering step of if-then-else rules. A further extension to SUMLOW could support modification of constraint rules via the user interface.

### *Single Stroke Recognition for Text*

The multi-stroke algorithm we used for UML diagram element recognition is unsuitable for text recognition as users have too much variation in their writing styles. We chose to use the well-known Rubine's single stroke algorithm in SUMLOW for text recognition [32]. In Rubine's algorithm, a gesture is an array  $\mathbf{g}$  of  $P$  time-stamped sample points:

$$\mathbf{g}_p = (x_p, y_p, t_p) \quad 0 \leq p < P$$

A vector of "features" is extracted from the input gesture (e.g. size of enclosing rectangle, average distance/time between points, etc):

$$\mathbf{f} = [f_1, \dots, f_n]$$

A linear discriminator is used for pattern recognition. The recognizer is trained from a number of examples of each gesture, typically around 10 or so. A Gesture class  $C$  has weights  $w_{ci}$  for  $0 \leq i < F$ , where  $F$  is the number of features.

Training determines the weights  $w_{ci}$  from the set of example gestures and a closed formula is used instead of iterative techniques. The formula is known to produce optimal classifiers under 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 then evaluated by a linear function over its features. The evaluations,  $V_c$ , are calculated as:

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

The classification of gesture  $\mathbf{g}$  is the  $C$  which maximizes  $V_c$ .

The main features of Rubine's algorithm are that: 1) it combines statistical gesture recognition with direct manipulation techniques; and 2) each gesture is recognized independent of its shape and size, but has to be drawn in a single stroke. While this algorithm has been used by many other researchers to develop single-stroke text (and shape) recognition tools, the use of a single-stroke algorithm for sketched text is also not ideal due to a need for considerable per-user training to provide acceptable recognition accuracy. A training module is provided to allow users to train the system to improve accuracy. We see this algorithm as an interim "convenient to implement" solution until the more robust algorithm used by Microsoft's Tablet PC extensions has a readily accessible API that can be accessed easily from SUMLOW.

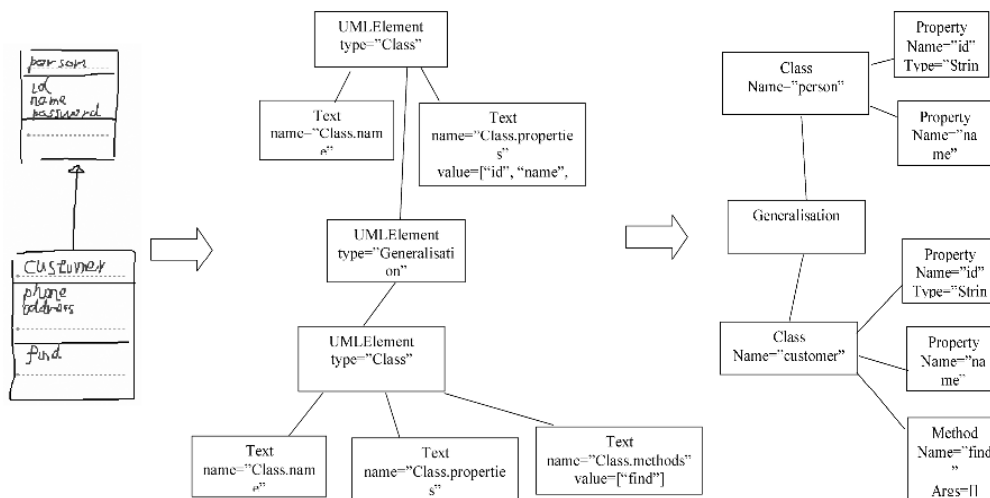
### *Formalisation of Sketched Content*

Formalised UML diagrams are produced by converting sketched content to corresponding UML diagram elements. In order to formalise a sketched diagram SUMLOW traverses the sketched diagram contents creating the formalised content. This process includes separating a single SUMLOW formalised design diagram into multiple diagrams where it contains elements from incompatible UML diagram types. The sketch formalisation process proceeds as follows:

1. Each sketched element that has been recognised as a UML iconic symbol has a formalised UML element created e.g. for a sketch element recognised as a class icon, SUMLOW creates a Class UML element
2. For each formalised UML element, each text sketch associated with its source sketch element are used to set values of the target UML element e.g. the text sketched over the class name annotation is used to set the Class.name attribute of the new UML Class element; the list of text sketches over attribute annotations are used to set the Class.attributes list of the new UML Class element
3. For each sketched element that has been recognised as a UML relationship symbol, the source and target UML elements and the relationship symbol type are used to create a new UML relationship between the source and target UML elements e.g. two Class elements connected by a sketch recognised as a UML association relationship have a UML Association relationship created between them
4. For each formalised UML relationship element any text sketches associated with its source sketch element are used to set values of the target UML relationship element e.g. the text sketched over the association name, parent arity and child arity annotations are used to set the Association.name, Association.parentArity and Association.childArity values for the next UML Association relationship
5. For each set of compatible UML diagram elements created , a new UML diagram is created and populated with the elements and relationships e.g. from a SUMLOW sketch containing UML Class and Object elements and Association and Invocation relationships, a UML Class Diagram containing the classes/associations and a UML Collaboration Diagram containing the objects/invocations are created. Invalid relationships between formalised UML elements e.g. a Note Link relationship between two Class elements, are removed from the resulting formalised UML diagram. UML Sequence Diagrams are treated as special case due to their complex layout rules – the user must specify they are sketching a Sequence Diagram (by drawing a horizontal line) and only compatible UML elements and relationships are converted to the formalised UML Sequence Diagram target.
6. Any unrecognised sketch elements are treated as informal secondary notation in the sketched diagram and are not converted into formalised UML diagram content. This includes any “dangling” relationships in the diagram i.e. without a source or target UML symbol.

Figure 16 illustrates this process for part of a UML class diagram sketch. The sketch drawn by the user (left) is converted into the sketch elements (centre), which denote recognised UML symbol type and associated text annotations on the symbol. UMLElements recognised as relationships are linked to the parent and child UMLElements of the relationship (using proximity to the start point and end point of the sketched line representing the relationship). During formalisation appropriate UML diagram elements are created (right) from the sketch data structure elements.





**Figure 16. Formalising part of a class diagram sketch.**

SUMLOW implements the UML diagram semantics that it uses during formalisation of sketch content as if-then-else rules in the formalisation component. These are not directly modifiable by end users. The formalised representation of UML diagrams is based on the UML 1.5 meta-model and SUMLOW saves and loads this from an XML format. Modification of formalised content is supported including moving/resizing elements and deletion. These modifications are applied to the sketched elements the formalised UML elements were derived from.

An export to XMI facility is provided which traverses the SUMLOW formalised UML diagram data structure to generate an XMI interchange format suitable for import by XMI-compliant UML tools. A Visitor implementation traverses the UML diagram data structure, as illustrated on the right of Figure 16, and converts this to XMI-compliant XML elements, attributes and text values. Currently SUMLOW does not support import of XMI format diagrams from such tools.

## Implementation

We chose to use a “LIDS” experimental E-whiteboard [1] that provides a back-projected opaque surface for displaying the SUMLOW user interface. A Mimio data capture device provides pen input for the application. We chose this technology due to its availability from other New Zealand-based researchers. In addition, this provides a low-cost and yet relatively high-performance E-whiteboard technology. Other E-whiteboard systems would be equally suitable to use with SUMLOW.

The SUMLOW application is written in VisualBasic and uses VB 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 sketch manipulation. One disadvantage of this approach is that no right-mouse button is supported, limiting some interaction styles (e.g. requiring use of pen-tap on modality buttons rather than in-context pop-up menus).

We implemented sketched shape and formalised UML data structures with VB object arrays. The sketched shape strokes were based on the timed strokes used in Rubine's algorithm but supporting grouping of multiple strokes into one sketched shape or line of text. The multi-stroke recogniser takes a set of strokes potentially representing a UML symbol and recognises each basic geometric shape within the set of strokes using a version of Apte's algorithm. A secondary filter implemented in VB applies our "drawing rules" approach to identifying a most-likely match to a UML symbol. The single-stroke text recogniser uses Rubine's algorithm and a training module to identify lines of text, each character written with a single stroke. The training module allows a user to specify ten examples of each character and this training set is saved to an MS Access database. We used a database rather than files as we found this easier for re-training during use of SUMLOW.

We used XML files to store saved SUMLOW sketched diagram data structures and formalised UML diagram data structures. We implemented save/load facilities using VB's XML support via the Microsoft XML Document Object Model (DOM) facility. A SUMLOW sketch diagram data structure is used to create a DOM object in memory and then the DOM is written to an XML file. Similarly, an XML file holding a sketch diagram is loaded into a DOM and then the DOM's Nodes are traversed to recreate the sketch data structure. The same approach was used to implement formalised UML diagram data structure save and load. An export facility was developed to create an XMI-compliant XML data structure in a DOM for export of a formalised UML diagram to another XMI-compliant tool.

In retrospect it would be better to use VB libraries to serialise the sketch data structure to an XML file and vice-versa. The formalised UML representation in memory should be based on the XMI standard and formalised UML diagrams written directly to an XMI-compliant file format, serving both as a persistent storage for SUMLOW and an export format to other UML modelling tools.

## **Discussion**

We carried out two evaluations of SUMLOW. In the first evaluation we have used the Cognitive Dimensions [14] framework to assess user interaction characteristics of sketching-based UML design compared to the same activities carried out using a conventional whiteboard and a conventional UML design tool. We carried out a survey of several industry-based UML designers – all highly experienced with using a conventional whiteboard and conventional UML design tools for both early phase and detailed software design with the UML – performing a sequence of collaborative, early-stage design tasks with SUMLOW. This evaluation provided user feedback on the tool's suitability for UML-based software design.

### *Cognitive Dimensions Analysis*

We carried out a Cognitive Dimensions (CD) assessment [14] to gauge the support of SUMLOW for exploratory UML design compared with conventional whiteboards and UML CASE tools. In this assessment we measured different aspects of diagramming with our SUMLOW prototype and compared these to similar features in conventional UML design tools and a conventional whiteboard UML design activity. Our CD-based usability results are summarised below.

- *Viscosity* measures a user's ability to change content in a visual language tool. Conventional whiteboards are to some degree very viscous – changing already drawn content can be very cumbersome e.g. moving and resizing requires erasure and redrawing. Many aspects of SUMLOW UML sketches require much less effort to change than in a conventional whiteboard e.g. redrawing over the top of a shape in order to resize it, selection and drag to move icons etc. Similarly, these tasks compare favourably with conventional UML CASE tools where both limitations of conventional mouse editing operations and UML design constraints sometimes interfere with user's ability to change diagrams. Other editing operations in SUMLOW require more effort e.g. movement of a shape is a sequence of change mode/select/drag vs click-and-drag in most mouse-based tools. In addition, many conventional tools apply layout and appearance heuristics to a design as the user builds it e.g. repositioning shapes to a grid automatically. This is of course contrary to the key idea of SUMLOW providing a sketching facility like a conventional whiteboard interface.
- *Secondary notation* measures the support for informal, user-defined notations e.g. layout or appearance in visual language tools. In SUMLOW the user has great freedom to sketch any type of secondary notation. This is more flexible than almost any conventional CASE tools we are aware of and in fact is similar to the infinite range of secondary notation available to users of a conventional whiteboard. However a consequence is that to recognise sketched diagram elements that are primary notation some drawing operation ordering and sub-element conventions must be met (the “drawing rules” for our multi-stroke recogniser). This can adversely affect usability of SUMLOW as it sometimes does not recognise an element and treats it as informal secondary notation. SUMLOW, unlike conventional UML tools, also allows users to mix notational elements, which other researchers have identified as useful in early phase software design. As SUMLOW uses no automatic layout algorithm for diagrams this allows users to position diagrammatic symbols as a form of secondary notation. Some conventional UML CASE tools enforce an automatic layout algorithm which may impair early design work in particular [7].
- *Provisionality* measures the degree of commitment to actions or marks. Is it possible to sketch things out when you are playing around with ideas, or when you aren't sure which way to proceed? Both the conventional whiteboard and SUMLOW provide excellent provisionality support, with no commitment required until sketches are completed. By contrast conventional UML tools require immediate commitment.
- *Premature commitment* measures constraints on the order of doing things. SUMLOW has some disadvantages over conventional whiteboards in that shapes must be drawn in the correct order for the recognition algorithm to work and some advantages in that initial layout commitments may be readily modified. The allowance of “dangling” relationships in SUMLOW provides an advantage over conventional UML tools.
- *Closeness of mapping* measures how closely a tool's notation matches the target domain's concepts in a visual language tool. A conventional whiteboard enforces absolutely no constraint on the appearance of UML designs drawn upon it, which is both an advantage and disadvantage. For example, it allows each user their own idiosyncratic shape drawing which provides flexibility at the cost of inconsistent appearance. As we aim to support UML modelling with SUMLOW then all sketched diagram elements must have some degree of similarity to computer-drawn UML elements in order to be recognised both by the recognition algorithm and also by the end user. This constrains the user to a degree and resolving mis-recognition can adversely impact on the usability of the tool. Conventional tools do not have

this problem as the user is forced to specify the exact type of element to add to a diagram by the tool's syntax-directed editing interface. Most conventional UML design tools enforce some UML syntax and semantics constraints during editing and studies have shown this to be quite problematic for exploratory design work [7].

- *Hard mental operations* measures how hard the user has to work in order to understand information or instruct a visual language tool. As conventional whiteboards enforce no constraints on users a (common) consequence may be very difficult to understand diagrams [22]. Similarly ambiguous or poorly-drawn sketches in SUMLOW can cause confusion for both the tool's shape and text recognition algorithms and its users. SUMLOW sometimes cannot recognise the information intended or makes mistakes and recognises things as the wrong construct leading to mismatches between the user's mental model and the tool's formal model. The ability to add any UML notational element to any diagram exacerbates this as the tool cannot assume a smaller set of allowable elements as in conventional, more rigid UML tools. The learning curve of the text and shape sketching/recognition algorithms currently make learning to use the tool in some respects more difficult than conventional mouse and keyboard-driven UML CASE tools. For example, users must draw complex, multi-stroke UML element icons in a small number of allowable orders else SUMLOW will fail to recognise them. While this has not proved to be overly problematic in practice, it does impose extra hard mental operations on the user, particularly novice SUMLOW users.
- *Abstraction* measures the ability to specify new abstractions that extend the notation. Conventional whiteboards allow infinite extensibility of notations. Both SUMLOW and conventional UML tools are quite constraining. However, SUMLOW does allow the text recogniser to be retrained offering some customisation support and does support an infinite variety of secondary annotations.
- *Hidden dependencies* measures the degree of hidden dependencies between information in the same or different views in a tool. Conventional whiteboards rely totally on users explicitly drawing dependencies or communicating them to each other via gestures and speech. In conventional UML design tools there are dependencies between the same UML model element shown in multiple views by element name only, without explicit visual linkages, but often with navigational support to mitigate the hidden dependencies. SUMLOW also represents dependencies between such UML elements internally without explicitly representing them in views. SUMLOW has additional hidden dependencies that don't exist in conventional UML design tools or whiteboards – the link between a sketch element and its recognised shape or text. These dependencies are represented by mouse-over hints in the SUMLOW sketch views and formalisation of a diagram uses them to explicitly represent sketch elements as computer-drawn shapes and text. The user can over-ride a recognition attempt by SUMLOW by selecting an element and manually indicating what type it is, or redrawing over the top of it to force update of its appearance and re-recognition by SUMLOW, changing the recognition hidden dependency.
- *Progressive evaluation* measures the degree of step-by-step evaluation of information supported by a visual language tool as it is used. Conventional whiteboards do not support progressive evaluation. The SUMLOW shape and text recognisers and the user-demanded formalisation of designs both support progressive evaluation. Shapes are recognised after the user has drawn multiple lines and pauses, or moves to a different part of the sketching area. Text is recognised after the user has sketched over the top of a text annotation and moves to a different area of the owning

shape or diagram. Design views are created and modified as a sketch view is manipulated and can be viewed at any time or simultaneously with SUMLOW sketch views.

### *User Evaluation*

We carried out a user evaluation of SUMLOW to assess its usability and appropriateness for early design UML diagramming. The selected evaluators of SUMLOW were first given a tutorial of how to use the tool and a chance to try out the tool for half an hour each with the tool designer present and answering any questions about using the tool. They were then given two simple design tasks – one single-user and one two-user collaboration. These tasks involved diagram sketching, manipulation and revision, and formalisation of design sketches. The evaluators were then asked to complete questionnaire on their experiences using SUMLOW for these tasks. 11 evaluators took part: 5 postgraduate computer science students and 6 analyst/programmers. Each of the industry-based analyst/programmers had between 2-5 years (average 3.6) working experience in software design.

The closed and open questions in our survey concentrated on:

1. Feedback on SUMLOW's utility for early design UML diagram production, and
2. users' assessment of the accuracy of the shape recognition component

Key survey findings are summarised as follows:

- Overall nearly all of the evaluators found SUMLOW was easy to learn and use. Almost all evaluators found SUMLOW's pen manipulation of diagrams to be very time efficient, especially for the early stage UML design tasks they carried out. Less than 25% of users had difficulty learning to use the moving and resizing operations of SUMLOW.
- The example design review and design revision tasks – not early-stage design tasks – that we asked evaluators to carry out were thought to be less suitable for SUMLOW. Ideally these tasks require more UML diagram syntax and semantics constraints to be checked while manipulations are carried out on the diagrams. However all evaluators could perform these revision tasks with SUMLOW sketches and over 70% felt SUMLOW sketch views provided a better environment to discuss and collaboratively revise designs than conventional UML design tools.
- All evaluators felt that good feedback is provided to users of pen manipulations of diagram content both while it is in progress and when finished. This includes active feedback to users of changes being made to diagram content and results of manipulations and the shape and text recognition engine running results.
- Almost all users felt the SUMLOW user interface provides a good design for early-stage UML design work. However nearly half of the evaluators found the SUMLOW modal interface to be difficult to use initially. However as this approach is close to that used by most conventional UML CASE tools nearly all evaluators found it acceptable after some experience with it. All evaluators noted that the SUMLOW interface provides many features not supported on a whiteboard. Some make diagramming easier e.g. automatic resizing and preservation of content, but others are less intuitive e.g. having to change drawing mode to select and delete items.
- Several users suggested more use of gesture-based operations in SUMLOW e.g. two lines drawn quickly through a shape or text to delete it instead of selection then deletion gesture. Currently gestures are used to initiate and action moving, deleting and front/back repositioning of shapes and draw-over for resize, which was found by users to be

intuitive and helpful. Other editing operations that may be better supported via gesture-based manipulation or draw-over include reconnection of lines from one shape to another e.g. by dragging one end point; moving text within a shape e.g. re-ordering Class icon attribute or method text; and resizing internal parts of a shape e.g. size of Class icon method compartment or Note icon diagonal line.

- Most evaluators stated that the ability to annotate diagrams in flexible ways is important and a useful improvement on existing CASE tools, especially when discussing designs and to capture design rationale
- All evaluators stated that SUMLOW's lack of enforcement of UML diagramming constraints during sketching was found to encourage exploratory design. Most evaluators stated that the ability to formalise a sketched UML design allows some standard UML diagramming constraints to be enforced periodically which was useful. Several evaluators wanted some ability provided to allow users to reconfigure the SUMLOW shape recognition and UML diagram constraint rules currently hard-coded into the tool. This would allow the user to "re-train" the complex shape recogniser in particular to allow different ordering of drawn sub-shapes. The ability of different users to enforce or relax particular UML diagram constraints would also be helpful. However none found the lack of this facility to seriously impact the usability of the tool for early design tasks.
- The text recognition component was found by almost all users to be unsatisfactory. The time to train the text recognition component, its lack of accuracy and the limitation to single stroke per character were all criticised by evaluators. This was found to be the major limitation of SUMLOW at present.

During user evaluation of SUMLOW we tracked the accuracy of the shape recognition algorithm by instrumenting the tool and determining when users over-rode the recognition algorithms. The shape recognition algorithm result was over-ridden by users either redrawing immediately over a shape to replace it and have the replacement re-recognised, or manually selecting UML shape using a pop-up menu. The results of tracking SUMLOW's shape recognition accuracy are shown in Table 2. Overall this shows acceptable performance for most UML components. The two items having the highest recognition rates are the Activation (91.7%) and Object (90.1%) icons. The reason appears to be because they are two simple rectangles which are easy to draw and recognise.

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

**Table 2. Accuracy of SUMLOW shape recognition.**

The item with the lowest recognition rate is the Node (69.2%) icon. This is probably due to its complexity. The second worst recognizable icon is the Use case (73.2%) which is bit surprising as it is just a simple ellipse shape. However, during the user evaluation experiments we observed that most users were much better at drawing straight lines (lines, rectangles, etc.) than curves (circle, ellipse, etc.), resulting in the simple geometric shape recogniser of Apte's algorithm identifying the former much more accurately than the later. This is an interesting finding and we may try to find solutions to address this issue in the future. SUMLOW does not currently use "contextual" information i.e. the last 2-3 UML shapes recognised which may greatly improve these rates e.g. if last shape was Actor, then next is very likely to be Use Case. Two limits with our drawing rule approach is that most users found it took time to understand the permissible sub element orderings required and users can not directly change these drawing rules for sub-element ordering.

A major qualitative problem identified by the evaluators was the poor performance of SUMLOW's text recognition. This required much effort to train and then performed poorly for most users. Text recognition can be over-riden by either redrawing the text or using an on-screen pop-up keyboard. Most evaluators resorted to using the pop-up on-screen keyboard in the end.

#### *Strengths and Limitations of SUMLOW*

From our evaluations of our SUMLOW E-whiteboard UML design tool we can conclude that it provides an efficient and effective sketching-based user interface on a large screen E-whiteboard. Both the design sketch elements and text are constructed and manipulated using pen-based input. The tool provides freedom from heavily-enforced UML modelling constraints and supports both flexible shape sketching and a flexible diagram and shape annotation facility. A pen-based diagram manipulation approach is used which enhances conventional whiteboard drawing manipulation facilities. The large display surface of the E-whiteboard hosting SUMLOW supports face-to-face collaborative design and multi-user manipulation of the sketched designs via turn-taking. Users can move between different UML modelling views, can use any UML modelling element in any view, and can formalise their designs periodically into computer-drawn UML

diagrams. These formalised diagrams can also be manipulated using pen-based input within SUMLOW. Users can export formalised design diagrams to conventional UML CASE tools for further refinement and use. In conclusion, these facilities within the environment encourage exploratory and collaborative UML-based early-phase software design. SUMLOW differs from conventional UML tools in its use of sketching-based diagram construction and manipulation. It differs from other sketching-based UML design tools in its preservation of hand-drawn sketches and emphasis on gesture-based manipulation of these hand-drawn, whiteboard-like sketches.

Disadvantages of our present SUMLOW prototype include recognition problems with some shape elements but particularly with the single-stroke text recognition algorithm. Users must draw complex UML model shapes sub-element by sub-element in a limited range of orders so that the multi-stroke recognition algorithm correctly recognises them. This makes the learning curve for some shape modelling difficult, but is a trade-off that is difficult to avoid when providing both generous secondary notation capabilities and good recognition of primary notation. Currently SUMLOW doesn't use the most recently recognised or located UML symbols to help the recognition of the next sketched UML symbol. Some diagram manipulation functions could be enhanced to make it easier to specify what user operation is required. Many users felt that this could be improved by employing hot-spots on shapes or pop-up menus instead of modal buttons which distract a user and slow their sketching down. Gesture-based instructions could also be used. One-directional translation of sketch to formalised UML model design is a limitation as it means users cannot import a UML diagram from a CASE tool and then use the sketching interface to review and modify it. There is no current support for distributed sketching-based design with multiple E-whiteboards, limiting SUMLOW's current use to a single E-whiteboard deployment.

#### *Future Work*

A number of enhancements can be made to SUMLOW to improve its usability particularly improved text recognition and shape recognition facilities. We are investigating using multi-stroke recognition systems that are trainable by end users for both shape and text recognition support [20]. These algorithms have been shown to be much more accurate than those currently employed in SUMLOW with the added benefit of per-user customisation support. In addition, we want to enhance recognition of shapes to take into account recently recognised and closely located symbols. We believe this will further enhance the recognition system. Currently we have only investigated use of a single E-whiteboard by one or more designers. We plan to include a generalisation of our sketching and E-whiteboard presentation support into a meta-CASE tool we are developing, making it much easier to "E-whiteboard" enable a wide range of design tools. This has the added advantage that the meta-tool specifies the visual language meta-models and constraints in a user-customisable and extensible form. These can be used during the recognition and formalisation processes rather than hard-coded model and rules as done in our current SUMLOW prototype.

We would like to explore distributed, same-time different-place collaboration support utilising multiple E-whiteboards. This will allow multiple users at a distance to interact with SUMLOW using their own E-whiteboard. Additional collaboration support facilities would be needed e.g. group awareness support and diagram element access controls. We are interested in deploying the tool on tablet PC environments to exploit their built-in pen-based input facilities and also to explore multi-user design via distributed tablet PCs as well as multiple E-whiteboards. Support for bi-directional sketch to formal UML model and formal model to sketch transformation would enable SUMLOW to be used



as a UML design review tool. The latter will 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.

## Conclusions

We have described SUMLOW, a sketching-based UML design tool for E-whiteboard technology. SUMLOW supports early-phase UML design using sketching-based input UML software design elements. These sketched elements are recognised proactively by SUMLOW but with hand-drawn shapes preserved. Users can collaboratively design with SUMLOW using a large E-whiteboard, can flexibly annotate sketched designs with their own secondary notation, and gesture-based manipulation of design sketches and single-stroke text recognition are supported. SUMLOW allows users to flexibly mix UML design constructs and users may on-demand have SUMLOW formalise their design sketches and have export them to existing UML CASE tools. A key novelty of our approach is the preservation of sketched diagram content and co-existing, consistent sketched diagram and formalised diagram views. We have carried out two evaluations of SUMLOW, one using the Cognitive Dimensions framework, the other using target end users. These evaluations have indicated that our approach is promising for the target goal of better-supporting exploratory, early stage collaborative UML design.

## References

1. Apperley et al (2002): Lightweight capture of presentations for review, In *Proceedings of IHM-HCI*, Lille, France, ACM Press .
2. Alvarado, C. and Davis, R. SketchREAD: A Multi-Domain Sketch Recognition Engine. In *Proceedings of the 2004 ACM Symposium on User Interface Software and Technology*, 2004.
3. 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.
4. 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.
5. Black, A., Visible planning on paper and on screen: The impact of working medium on decision-making by novice graphic designers. *Behaviour and information technology*, 1990. **9**(4): p. 283-296.
6. 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.
7. Brooks, A. and Scott, L. Constraints in CASE Tools: Results from Curiosity Driven Research, In *Proceedings of the 2001 Australian Software Engineering Conference*, Canberra, Australia, 26-28 August 2001, IEEE CS Press, pp. 285-296.
8. Chen, Q. Grundy J.C. and Hosking, J.G. An E-whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams, In *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments*, Auckland, New Zealand, October 2003, IEEE, 219-226.
9. Damm, C. H. Hansen, K. M. Thomsen, M. Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard. In *Proc CHI 2000 on Human factors in computer systems: the future is here*, ACM Press, pp. 518-525.
10. Donaldson, A. and Williamson, A. Pen-based Input of UML Activity Diagrams for Business Process Modelling, In the *HCI 2005 Workshop on Improving and Assessing Pen-based Input Techniques*, Edinburgh, Scotland, Monday 5 September 2005.

11. Fowler, M., Scott, K. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 2<sup>nd</sup> Edition, Addison-Wesley, August 1999.
12. Goel, V., Sketches of thought. 1995, Cambridge, Massachusetts: The MIT Press.
13. Goldschmidt, G., Serial sketching: Visual problem solving in design, The backtalk of self-generated sketches. *Cybernetics and Systems*, 1992. **23**: p. 191-219.
14. 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.
15. Gray, J., Liu, A. and Scott, L. Issues in software engineering tool construction, *Information and Software Technology* 42 (2), February 2000, 73-77.
16. Grief, I., Seliger, A. and Weihl, W. A Case Study of CES: A Distributed Collaborative Editing System Implemented in Argus, *IEEE Transactions on Software Engineering* 18 (9), September 1992, 827-839.
17. Grundy, J.C. and Hosking, J.G. Engineering plug-in software components to support collaborative work, *Software - Practice and Experience*, Vol. 32, No. 10, August 2002, Wiley, 983-1013.
18. Hammond, T. and Davis, R. Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams, In *Proceedings of the 2002 AAAI Spring Symposium on Sketch Understanding*, pp.59-68. Stanford, California, March 25-27 2002.
19. Hse, H. and Newton, A.R. Sketched Symbol Recognition using Zernike Moments, In *Proceedings of the 2004 International Conference on Pattern Recognition*, Cambridge, UK. Aug. 2004. pp. 367-370.
20. Hse, H., Shilman, M. and Newton, A.R. Robust Sketched Symbol Fragmentation using Templates, In *Proceedings of the 2004 International Conference on Intelligent User Interfaces*, Funchal, Portugal. Jan. 2004. pp. 156-160.
21. Ideogramic, Pervasive UML, <http://www.ideogramic.com/>.
22. Iivari, J. Why are CASE tools not used? *Communications of the ACM*, vol. 39, no. 10, October 1996, 94-103.
23. Landay, J. A. SILK: sketching interfaces like crazy. In *Proceedings of CHI'96 on Human factors in computer systems: common ground*, ACM Press, pp. 518-525.
24. 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.
25. 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.
26. Mackay, D., Biddle, R. and Noble, J. A lightweight web based case tool for UML class diagrams, In *Proceedings of the 4th Australasian User Interface Conference*, Adelaide, South Australia, 2003, *Conferences in Research and Practice in Information Technology*, Vol 18, Australian Computer Society.
27. Mimio® home page: from <http://www.mimio.com> (Last Visited Tuesday, March 26, 2003).
28. 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.
29. 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.
30. Quatrani, T. *Visual Modeling with Rational Rose 2002 and UML*, 3<sup>rd</sup> Edition, Addison-Wesley, October 2002.
31. Robbins, J. and Redmiles, D. Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML, *Information and Software Technology*, 2000.
32. 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.

33. Shuckman, C., Kirchner, L., Schummer, J. and Haake, J.M. 1996. Designing object-oriented synchronous groupware with COAST, Proc of the ACM Conference on Computer Supported Cooperative Work, ACM Press, November 1996, pp. 21-29.
34. Smart Technologies Inc. (2002): *SMART Board*, [www.smarttech.com](http://www.smarttech.com).
35. Stahovich, T.F., The engineering sketch. IEEE Intelligent Systems, 1998(13 (3)): p. 17-19.
36. Sezgin, T.M. and Davis, R. HMM-Based Efficient Sketch Recognition, In Proceedings of the 2005 International Conference on Intelligent User Interfaces, New York, January 9-12 2005.
37. UML™ Home Page, OMG (Object Management Group), available from <http://www.uml.org/>
38. Volda, 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.
39. Wong, Y.Y. Rough and ready prototypes: Lessons from graphic design. in Human Factors in Computing Systems CHI '92. 1992. Monterey.