

# Contents:

Contents:.....	1
.....	1
Scribbles:.....	2
A Sketch Based Case Tool For .....	2
Software Development.....	2
1Introduction.....	2
2Background Research.....	3
2.1CASE environments.....	3
2.2Sketch tools.....	3
2.3Mind Maps.....	3
3Scribbles Concept.....	4
3Specification.....	4
4Design.....	5
5Implementation and Testing.....	5
5.1Paper like interaction.....	5
5.1.1'scribbles' in context.....	5
5.1.2Shape Recogniser.....	5
5.2Graph to represent vector Image.....	6
5.3Selection, Drag and Edit selected item.....	7
5.4UML rendering and generation.....	7
5.5Editing UML.....	8
5.6Code viewer and key word parser.....	8
5.7Code Generation.....	9
5.8File Input/output.....	9
6Evaluation.....	9
6.1Group Evaluation.....	9
6.1.1Problem One: snakes and Ladders.....	10
6.1.2Problem Two: Virtual Kitchen.....	10
6.1.3Results and conclusions.....	10
6.2 Individual Evaluation.....	10
6.3Resultant development.....	10
7Testing.....	11
8Summary and Conclusions.....	11
9Future Plans.....	11
9.1Short term plans.....	11
9.1.1Cut, Copy and paste.....	11
9.1.2Undo.....	11
9.1.3Requirements gathering.....	11
9.2Long term plans.....	11
9.2.1Full synchronisation.....	11
9.2.2Different languages .....	12
9.2.3Different models .....	12
9.2.4Export class diagram as XMI .....	12
9.2.5Gesture recognition .....	12
Acknowledgments.....	12
References.....	12

# Scribbles: A Sketch Based Case Tool For Software Development

**Christopher J Martin**  
**Final Year Project**  
**BSc (Hons) Applied Computing**  
**University of Dundee, 2005**  
**Supervisor: Dr Glenn Rowe**

*Abstract – CASE (Computer Aided Software Engineering) tools have been used to assist the software development process from the beginnings of software development. As computers became more powerful and complex, the size of applications running on them grew. This gave rise to the development of more powerful CASE tool to support the generation of these large applications. Modern CASE environment are themselves large complex programs. With this complexity comes a problem. The complexity of these CASE environments means that much of the early stages of design are done on paper and potential not taken through to later stages of design. Scribbles represent a simple CASE environment which provides the development team a means to create and maintain UML (Unified Modelling Language) class diagram in a way that is attainable from very early in the design stages. The entire process from deciding on how to conceptually model the problem through to code generation is supported.*

## 1 Introduction

Complex CASE environments provide a huge amount of functionality. This leads to a complex interface which is not conducive to supporting the creative processes. A second less pronounced problem is the level of complexity involved in the model being produced. A fairly standard method of modelling class architectures is in UML (Unified Modelling Language) class diagrams. Class diagrams provide a visual representation of how classes 'fit' together to produce the final solution.

The first problem of interface complexity is answered by making Scribbles focus on a precise problem, the task of designing class architectures. As this is a small subset of UML it immediately reduces the complexity. The second part; making Scribbles usable at the design stage, involved making the interface very intuitive and simple. Although the software developer is among the most adept at traversing great depths of menus and handling multiple non modal dialogues, the goal was not to produce a tool that could be just 'used' but one that was a joy to use.

There is also the question of cognitive load impeding the creative processes involved in conceptual design. As Scribbles aims to capture the early stages of design the cognitive load encountered must be low.

The second problem of trying to develop a complex model is addressed by adding a layer of abstraction. I looked to common practices in early design. Brainstorming is possibly one of the earliest possible design strategies. Brainstorming involves taking a catalyst generally something core to the desired solution and producing associated thoughts or themes. This is done in a group environment and no judgment is passed on generated ideas until the 'storm' is over.

As this is a common practice it seemed a natural starting point. The problem was to give some order to this collection of ideas and themes, and possibly guide the process. As this information is generally hierarchical in structure a mind map [4] offered the correct balance of structure and freedom to change and add. This concept will be explored later.

A hybrid mind map is used to describe the structure of the object model. This mind map is used to generate a UML class Diagram. The UML class diagram can be edited to fully describe the architecture of the final program. Java source code can then be automatically generated and viewed within scribbles.

## 2 Background Research

This section explores the markets best offering followed by some active research into sketch based case tools and finally into some more general applications of sketch based interfaces

### 2.1 CASE environments

The Rational Suite represents a professional CASE environment. I will focus on Rational Rose as this is the tool for creating and maintaining UML diagrams and documentation. This tool is extremely powerful and complex. It supports the entire UML vocabulary.

With this power comes complexity, from one selection event i.e. button click or mouse selection there are 171 different possibilities. This doesn't include sub menus or right click menus, many common tasks require multiple selection events. Couple this with four active panels and the scale of this Application becomes apparent. For a small development team there would need to be considerable value attached to modeling programmes in such a tool.

The whole motivation for UML is adding value to the development process. Producing and maintaining documentation that supports the development process. Complex tools such as Rational Rose are economically viable where large teams of interdisciplinary experts are involved and using UML to it's extent is a valuable process. Unfortunately small firms do not have the time or money to embrace such tools.

Another problem with these large tools is they are oriented around the problem and not the User. This is evident in the lack of support for conceptual stage of design [8]. There are a number of projects underway to try and combat this problem.

The Knight project [9] is being carried out within the Centre for Pervasive Computing at the Department of Computer Science, University of Aarhus. It aims to provide support for conceptual design. The interface is designed for use on a e-whiteboard. It uses gesture recognition to generate UML diagrams directly. This project has been running for some time and is defiantly representative of the cutting edge of user centred object modelling tools. This tool has reached the market via Ideogramic [10] a 'spin out' company.

This tool supports direct generation of UML via the e-whiteboard. Recognition of 10 gestures is used is used to provide fairly full generation and editing of complex UML diagrams. This is also supported by standard menu driven interaction. The speed of the recognition is excellent. The accuracy is not so great for large shapes. To define association types the end piece must be drawn, I found this almost impossible to get the correct gesture. There seemed to be a great deal of overlap in the classification space between the three gestures.

Another novel feature of this system is its wheel menus. Rather than the standard linear right click menu a wheel centred on the click position is displayed. This offers around three to six options all equidistant from the click positing. This removes problems of menu order and forces a small number of menu items.

### 2.2 Sketch tools

The technology to support 'paper like' interaction with a computer has been around for a while in the form of graphics tablets, tablet PC's and e-whiteboards. There are an increasing number of tools available nowadays that take advantage of 'paper like' interface paradigm.

A similar tool to Scribbles is ASSIST from MIT Artificial Intelligence Lab. Mechanical Engineering has many parallels to its software counterpart. The early stages of a mechanical engineering problem are also often solved on paper as CAD tools are cumbersome and offer no support to the conceptual stage of design [8]. ASSIST provides the ability to sketch 2D mechanical systems without defining what components are. Assist then provides the ability to simulate the mechanical system.

One of the aims of this system was to provide the aforementioned functionality without forcing the users to adapt the drawing style. This is a concept captured to some extent in Scribbles.

### 2.3 Mind Maps

Mind Mapping was developed by Tony Buzan in 1970. It is technique that has been used for studying and planning. It uses a combination of graphical and textual components. A Buzan mind map has a few key rules to it's production. A large landscape sheet of paper is used. They are strictly hierarchical based around a single stimulus generally a picture or sketch. Themes and ideas radiate out from this central idea or concept. Colour is a key way to denote linked themes. Use of pictures and sketches is also encouraged.

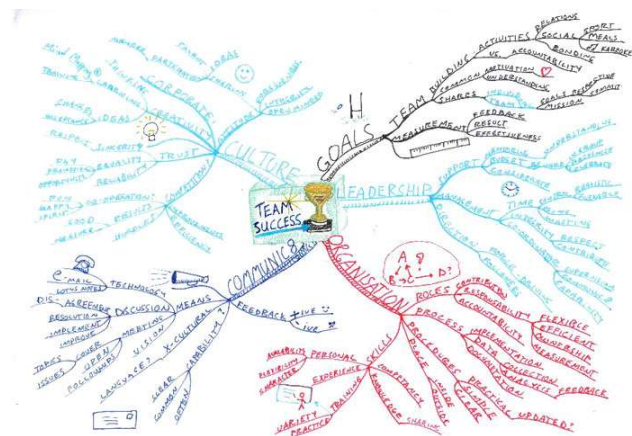


Fig 1 Mind Map

Research was carried out at Barts and the London school of Medicine and Dentistry, to determine the efficiency of mind mapping as a study technique [4]. Fifty students were given a 600 word passage to read, the passage was carefully selected to avoid prior knowledge of the subject. To determine a base line all students were tested. The group was then randomly split into two groups, mind map group and self selected study group.

These groups were given the passage again to study, with their given technique. The mind map group were taught how to mind map. When adjusted by the baseline the mind map group showed 10% increase in accurate recall of the passage. There was less enthusiasm for the mind map techniques which does present an interesting challenge.

An extensive study was carried out at the Loyola College in Maryland, Baltimore to find out if mind mapping proved valuable for their Executive MBA course. This paper, Mind Mapping in Executive Education: Applications and Outcomes [7], proves to be a valuable resource.

One of the key features of mind maps is the use of the whole brain. Physiological psychologist Robert W Sperry carried out pioneering work to develop the model of the split-brain. According to Sperry's work The two sides of the brain have different roles, generally the left cortex is concerned with numbers, words, reasoning, linearity and analysis where the right cortex deals with rhythm, images, colour, face recognition and imagination. Later it was discovered that each cortex contained some of the functionality of the other cortex. These observations were made possible by observing subjects with a severed Corpus Callosum the connecting bunch of fibres between the right and left hemispheres of the brain. Buzan [3] puts forth arguments that Einstein and Picasso used both sides of their brain in producing the finest work.

This concept of crossover between the creative and analytical centres of the brain is captured in mind map. This produces a fuller way to explore ideas than text alone. This study concluded that mind map was indeed a successful tool for Executives. There seemed to be no problems of motivation.

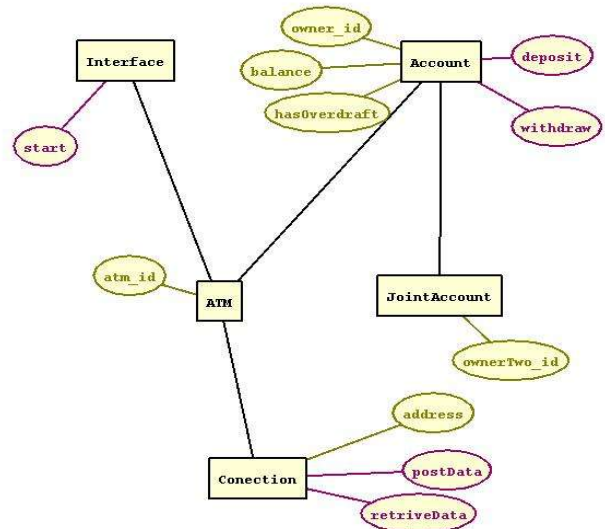
### 3 Scribbles Concept

There is a huge variety of CASE tools out there but their adoption seems limited. There are aforementioned tools which have taken advantage of pen based computing technologies this is without a doubt a step in the correct direction. For a tool to fully support the development process from brain storm to end product there must be a

stepping stone between ideas and the model that best represents them.

Scribbles has taken some of these key concepts of mind mapping through to a system of mind mapping Object Oriented Models (OOM). The initial phase involves documenting a brainstorming session in a mind map. At this stage colour is used to connect themes. The scribbles software is used on a tablet PC that supports freehand sketch input of this mind map stage.

The second phase is to modify this initial mind map into a diagram representing an OOM. This is done by defining nodes on the previous map as *class*, *method* or *variable*. Nodes that aren't used can be deleted or kept as a memory aid. This diagram represents the architecture of the program.



Generation is transparent. As a simple subset of UML is supported it is relatively straightforward to inform the developer how the UML impacts the code structure. This ensures that the developer gets the code they are expecting.

### 3 Specification

With a good knowledge of the tools on the market and a concept for an improved tool it was necessary to refine this concept into an application. This entire problem space is modelled in the Use Case Specification [appendix A]. Use cases are used to model a problem at high level, based around the user interactions the system will support. A single Use Case can be seen as a considerable chunk of the problem. Each use case is further described as a dialogue between actors and the system. This is to gain an understanding of how the user will interact with the system from an early stage.

This more focused picture of user interactions was used to assist in the development of requirements. Requirements are a detailed list of what the system will do. They are split

into to main categories; functional and non-functional. Functional requirement include actual functionality provided by the system. Non functional requirements include things related to how these tasks are performed, with relation to performance and system requirements.

Requirements Gathering is generally an extremely challenging process, as clients or users tend to not be computer developers. In this case I was rather lucky as I could base my requirements around my experiences of computer modelling packages and ultimately develop a tool I would use.

The requirement have a; *should, shall, may* structure, this outlines the level of importance with should being core functionality, shall; hopefully included and may; if time allows. For the full requirements document please view [appendix B](#)

I have adopted an iterative unit based approach to managing this development. This is based around many common extreme programming [11] techniques. As this was a fairly short term project with only one developer the normal length of an iteration [appendix C](#) was one week. An iteration consists of a period of design followed by a period of implementation and finally testing. In parallel to this is a Architectural Planning and refactoring stage. This is intended to manage the ‘bigger picture’ making sure a quality, open architecture is achieved. Each new iteration starts with a meeting which reviews the previous unit’s successes and looks forward to decide on the next unit.

One key feature was take from the SCRUM [12] methodology, is that each unit shall provide a visible, demonstratable piece of functionality. I feel this is very important; it causes units to be concise and only do things they need to. It also makes reviewing progress easier. In a real world scenario where the client may not be a developer it will be possible to run acceptance tests at the completion of each unit. There is in essence no need for long winded reports on where the project is; a demonstration of functionality provided so far will fill this task.

## 4 Design

This project has been based around week long iterations of design implementation and review. For this reason detailed information about algorithms and function specific design decisions have been placed in the next section. This section discusses the architectural structure that has been arrived at.

Scribbles has a fairly open architecture. Core parts of functionality are distributed in loosely coupled classes. One key feature of this architecture is the graph structure (class:- Diagram) used to store information on the various diagrams. This stores enough information to make rendering UML, code or a diagram possible. This was a

critical design decision as maintaining different structures; to represent the different views would prove a concurrency nightmare.

Structural Information is added to this Diagram via the DrawingPanel which supports use of free hand input. This is contained in the interface which allows finer details to be added to diagram Elements. The ability to view code is delivered by the Editor class which contains tabbed CodePanels. A CodePanel is based around a JTextPane which allows different styles to be applied to different sections. This allows comments to be coloured green and key words blue.

## 5 Implementation and Testing

### 5.1 Paper like interaction

The system is aimed to be delivered on a tablet PC. The purpose of the tablet PC is to allow ‘pen and paper’ like interaction. This involves some shape recognition techniques and also some simple interpretation of ‘scribbles’ in context.

#### 5.1.1 ‘scribbles’ in context

Before a shape recogniser event is triggered there are a few simple interpretations that can be made. Natural connection of shapes is facilitated by drawing a line from one shape to another. This is interpreted as a connection between the shapes. The interpretation is carried out as follows;

```
mouse_clicked_event(){
    If( mousePosition is in shape)
        startPoint = mousePosition
}

mouse_released_event(){
    If( mouse Position != startPoint &&
        Mouse Position is in shape)
        Add connector ( startPoint,
                        mousePosition)
}
```

Fig 3 connector algorithm

This approach is also taken for deletion of an existing object. If an input scribble is completely bounded by a single existing shape, the shape is deleted. This is to imitate scoring out an incorrect part of a pen drawing. This is extremely quick and easy to use. It facilitates quick recovery from errors, and keyboard free deleting.

#### 5.1.2 Shape Recogniser

The recogniser uses an algorithm based on template matching [12]. Template matching involves having a

template of the pattern you desire to recognise. This is placed on the input image in a number of places, to determine if this pattern exists in the input image. A simple method of calculating how well the template matches it's position on the input image is to perform a difference. In short the position on the input image that has the smallest difference to the template, is the most likely place the pattern represented by the template occurs.

The main problem with pure template matching is deciding where to place the template and at what orientation. In the crudest solution you could place the template at all positions in all orientations. This is hugely inefficient to the point of not being possible. My algorithm overcomes this by limiting the search window, and only looking for horizontally oriented shapes.

When the DrawingPanel detects a closed shape it sends the shape as a GeneralPath [14] to the recogniser. This means the recogniser can easily determine the size and position of the input shape as a GeneralPath implements the java shape interface and provides a GetBounds method. This allows the recogniser to generate templates of the correct size. The initial recogniser rotated these templates by +/-15° about the horizontal to allow for slight variation in real world input. This proved expensive for large images and didn't have much effect on the accuracy.

The other problem with template matching is that matching a generated geometric shape to a hand drawn image will never match well. The scribble drawn is 3 pixels wide if a 3 pixel wide template was generated there is little chance of getting good matches. To counteract this I generate templates with larger widths. This in essence gives a fudge factor also achievable by small amounts of scale, translation and rotation. As I am making a discreet classification of rectangle, ellipse or unknown this is sufficient. The actual orientation and exact position is not too important.

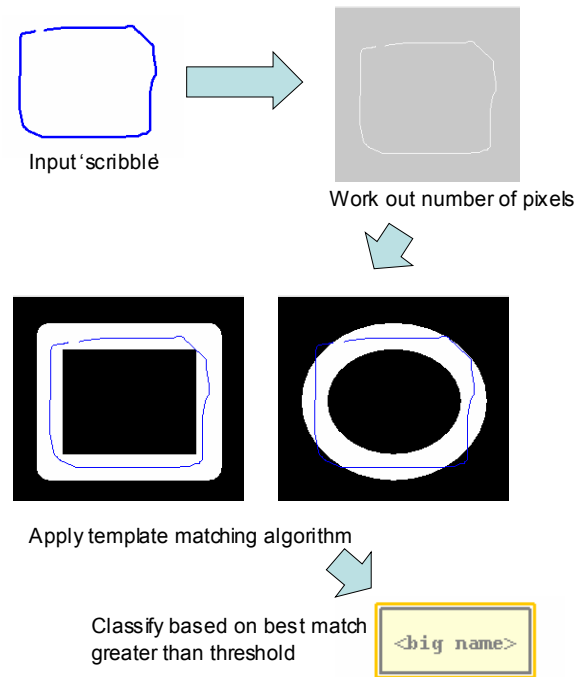


Fig 4 Shape Recognition images

This has a problem, if the template thickness is too large there is little distinction between rectangles and ellipses of a small size. To compensate for this the template thickness is proportional to the size of the template. This algorithm is fairly acceptable.

## 5.2 Graph to represent vector Image

As mentioned the structure used to store the diagram information is a graph. This models closely the real world structure of the diagram, as a non hierarchical mind map is essentially a graph. This graph is made up of nodes, in this case DiagramElements and edges in this case, connectors. This implementation consists of two linked lists, one for Diagram elements and another for connectors. The connectivity of the graph is represented by the Connector list.

This is convenient as many tasks don't utilize the graph structure, for example rendering the diagram. For this task the structure can be thought of as two independent Lists. This avoids complex and potential expensive traversals where they are not necessary. This also simplifies Graphics issues such as ordering and overlapping. The connectors are always rendered first, from the centre of each node, then the diagram elements can be rendered over the top. This gives consistent 'magnetic connections' unlike power points anchor points round the edge of shapes, which have undesirable effect when shapes are moved a large amount .

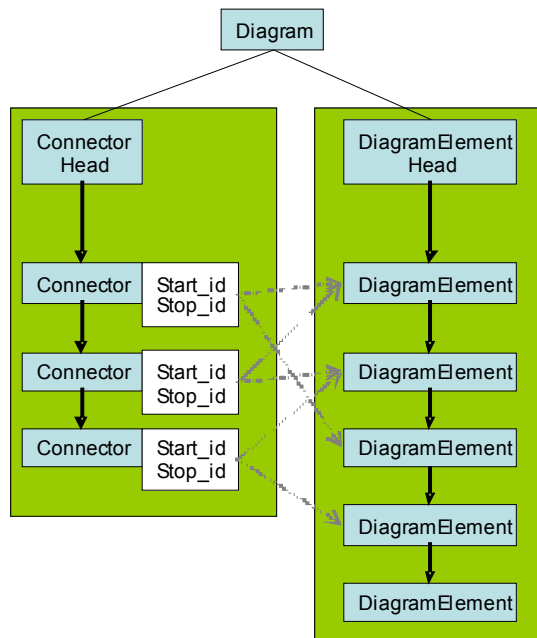


Fig 5 Graph data structure

The graph structure is used for enforcing good relationships, for example a method cannot be connected to more than one class. This validation is done by checking that methods and variables are leaf nodes in the graph, if they are not, the last connection(s) added is/are removed. This can produce some interesting results. When an elaborated diagram has nodes defined as class, method or variable it can seem unclear why connectors are removed. This is unfortunate but without knowing what the user wants to do, this is the best way to force structural correctness at this early stage.

The Graph structure is also used extensively for code generation. This will be explored in more detail in a section 7.1.

The responsibility for rendering the visual representation of the diagram is distributed to the individual elements. The concept being the 'image' in this case a `BufferedImage` [14] is passed around leaving the responsibility of actual rendering the 2D graphics to the individual nodes.

When the text in a `DiagramElement` is edited, the box or ellipse surrounding it auto resizes. This is done with a simple calculation which involves calculating the size of the box that would exactly contain the text. This can then be padded out by a fixed amount to form margins. The position the text is plotted at is described relatively from the centre of this new box and the centre of the text. This causes boxes to grow about their centre. Much of this kind relative positioning and auto centring was taken forward and used in the rendering of UML.

### 5.3 Selection, Drag and Edit selected item

The ability to drag and drop is facilitated largely through the use of the graph structure. The `DrawingPanel` allows the user to select items by left clicking on them and to drag them by holding down the control button and dragging the mouse with the left button depressed. This is only possible as some kind of vector based graphics are being used. The graph allows the `DrawingPanel` to check if a selection event is over a selectable item, if it is the item is set to selected. If the mouse is dragged and an item or items are selected the selected item(s) position(s) is/are updated, the screen is cleared and the diagram re rendered. This happens for every mouse event.

To make the connector 'magnetic' i.e. connect the same nodes when these nodes are moved about, they have no position information, just the node id of their start and end node. When they are drawn they are given the list of `DiagramElements` allowing their start and end positions to be retrieved this means when dragging takes place only a single position has to be maintained.

There are Boolean flags in the `DrawingPanel` to note whether one, none or many items are selected. The interface has a timer which poles the `DrawingPanel` to detect changes in selection. This allows the interface to load a newly selected item into the properties panel, and give the user the ability to edit it. This is rather complex as there are a multitude of selectable items and several different panels for editing them. When there are no items selected the properties window is blank as there is nothing to edit. To maximize drawing space it was decided to make this window auto hide and auto show when an element is selected. This also provides a cue to the user that the item selected is connected to the properties panel.

### 5.4 UML rendering and generation

Drawing the UML class diagram was a significant challenge, as on the one hand the connectivity of the graph is used to work out the structure and content of the UML classes, and on the other hand all the UML has to be drawn from simple 2D graphics which react to change in properties. The algorithm is as follows;

```

for each connector
  if( start_node.type == CLASS &&
      Stop_node.type == CLASS)
  {
    render connector
  }

for each DiagramElement
  if( thisNode.type == CLASS){
    get member methods
    get member variables

    render class box
    render icon + class name
    for each method
      render icon + method string

    for each variable
      render icon + variable
string
}

```

Fig 6 Draw UML algorithm

The first stage of rendering a class involves gathering all the components such as variables and methods. This allows the dimensions of the class's bounding box to be calculated. The height and width of a single character are defined as constants and are used to allow relative coordinates for lines and strings to be plotted at. Implementing this was a rather tedious process but allows the UML diagram to auto resize edited .

To follow best practice the standard variable, method and class icons are also used. This helps form association between the diagram and UML stage as well as making the UML more readable. The placement of these icons is also relative.

Rendering simple associations (relationships in UML) is simple however for UML to describe class structure, more complex associations are used, these take the form of different end pieces. For the same 'magnetic' connectors to be used in the UML view, the point that a connector intersects the class box is needed along with the orientation of the line relative to the class.

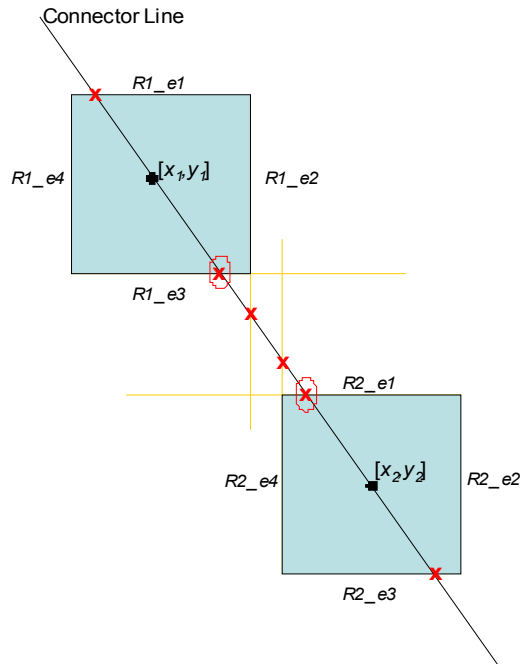


Fig 7 connector class intersection

The two rectangles widths and heights are also know, the rectangles are considered as four lines. First the 'inward' facing lines must be determined. In this case;  $R1_{e2}$ ,  $R1_{e3}$ ,  $R2_{e1}$  and  $R2_{e4}$ . This is done by working out which two lines lie between the two centre points.

As the lines that make up the rectangles are either horizontal or vertical they have equation  $y=k$  and  $x=k$  respectively. To find the point of intersection we use equation (1), where  $m$  is the gradient of the connector,  $(x,y)$  is a point on the rectangles edge and  $(x_1,y_1)$  is a point on the connector.

$$y-y_1 = m(x-x_1) \quad (1)$$

Equation (1) can be transposed to give Equation (2) and (3) which can be used to find the  $x$  and  $y$  value of the point of intersection given the equation of the rectangle edge.

$$x = x_1 + (y-y_1)/m \quad (2)$$

$$y = m(x-x_1) + y_1 \quad (3)$$

To find the correct point of intersection that lies on the rectangle edge, rather than it's extrapolation, the value is checked against boundaries of the given edge.

The orientation of the connection line is give by the arc tan of the gradient of the connector.

## 5.5 Editing UML

It was decided early on that non modal dialog boxes were to be avoided when editing the UML class diagrams, mainly for the reason that they remove attention from the

diagram and to fully edit a class would require tabs or a fairly loaded panel. To keep the focus on the diagram and the content of the properties panel down editing is facilitated by direct manipulation. To edit an item on the class diagram click on it and it is loaded into the properties panel. There are three possible items that can be selected; a class, method, variable or association. Each results in a different properties panel.

As this is a custom drawn diagram the selection process was implemented manually rather than with buttons. A similar process was carried out as when the UML is rendered. Each class diagram element contains a method called 'getClassBounds' this method carries out the same work as the renderUML method, it gathers all the methods and variable, and calculates the dimensions of the classBounds. This is used for a preliminary sweep to see if any classes have indeed been selected.

If a class has been selected a further refinement is carried out to find out which element has been clicked upon. The number of elements a class has can be calculated easily by subtracting the space required for the title box and then dividing the remaining number by the height of an attribute. Once this is done each attribute space is checked to determine whether the click was over it. After this stage is carried out and the attribute that has been clicked on is known all that remains is to find this attribute in the graph and set it to selected, the Interface notes the selection event and the new item is loaded into the appropriate properties window.

Another challenge here was to provide a quick and easy way of editing the relationship types in the UML. Unlike most systems the relationships are defined as simple associations in the diagram stage. These then need to be edited to form UML associations. As mentioned a subset of relationship types are supported. There are six types in total. As some of these relationships can run in both directions e.g. aggregation could be on either end of a relationship, it was decided the easiest way to support this was to offer all possibilities on the list. This allows any errors to be recovered from easily.

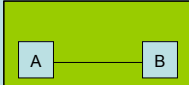



Diagrammatic	Definition	Effects on Code
	A and B are logically related	No implications
	A inherits B	A inherits B
	B aggregates A	B contains a single variable of type A
	A aggregates many Bs	A contains an array of type B

Fig 8 UML relationships

The actual delivery of the list of connectors is provided by a custom combo box using large icons to depict each relationship type. When selected it shows the three main relationship types, aggregation, multiple aggregation, and

Inheritance in both directions. These are placed at the start of the list as they have impact on the code and this tool is targeted at the more code oriented developers who often would not consider object modelling.

## 5.6 Code viewer and key word parser

To continue with this seamless progression from mind map to code it seemed wise to provide a code viewing facility within scribbles; as well as the ability to output source code files. For this implementation the code generated is java. The ability to view the code in scribbles strengthens the association between model and code. This is important as scribbles hopes to increase the value associated with object modeling. It also allows the developer see how changes in the model impact the code immediately.

The code viewer is based around what you would hope to find in an editor. The class files are displayed in separate panels within a tab pane. Key words are coloured blue and comments green. An artifact of the model generated code is the fact that a variable doesn't necessarily have to have a type associated with it, this is the same with method return types. When code is generated '<not\_defined>' is used in place of type. These terms are coloured red to highlight the need to define types.

There is no standard way to change the colour of given words in a java text component. Simple Java text components do not support multiple text styles so a JTextPane was used. A parser was written that returns the start and end positions of key words, comments and '<not\_defined>' tags.

```

For each character
  if(thisChar == terminator)
    For each key word
      If(temp == thisKeyWord)
        Index[ptr++] =
position -
                thisKeyWord.length
position
                Index[ptr++] =
                initialize temp
      else
        temp = temp + thisChar
    position++

```

Fig 9 key word parser algorithm

This algorithm is applied to a character array retrieved from the JTextPane. A terminator refers to any character that can come directly before or after a key word. The Index array is an integer array of start and end positions, this is returned allowing the key words to be coloured.

The algorithm used to find comments is very similar, in this case the same set of terminators is used to 'open' and 'close' a key word. For comments a different 'opening' and 'closing' is used e.g. '/' followed by '\*' signifies the opening of a multi line comment, and '\*' followed by '/' signifies the close. This system is by no means optimal. Parsing the entire file for every key press is acceptable for around five hundred lines of compact code. There are three sweeps being made through the code for, comments, key words and '<not\_defined>'. This could be combined into one more complex 'pass' this was not necessary to satisfy the requirements as this section is for class architecture viewing and minor edits, even a bloated complex class is unlikely to extend beyond two hundred lines. Further refinements could be made by parsing the area local to the key press.

This was originally hoped to be developed into a simple editor. As it is possible to compile and run java .class files from the command line it wouldn't be to challenging to build in the ability to compile and run code. It was decided not to go down this road as the functionality provided in even a simple editor far extends beyond running and compiling code. The likelihood of a development team using the limited editor functionality that could be provided is small.

## 5.7 Code Generation

Generating code from UML involves two stages. You must 're-draw' each class as code rather than a class on a diagram; this is relatively straightforward as this is very similar to drawing the classes. The more complex task is using the relationship types and connectivity of the graph to determine inheritance relationships and also implications of aggregation.

The whole point of UML is that it is abstract from the implementation, 'Scribbles' separates the structure and the actual code generation. The structure and content is supplied by the graph. The actual syntactically correct code is generated by a separate JavaGenerator class. This means implementing code generation into a different language a fairly easy task.

The Graph has a couple of methods; getInheritance() and getAggregation(). These take the id of the class being generated as an argument. The connection list can then be stepped through querying for either inheritance or aggregation relationships involving the given class. For aggregation the relationship name is used as the variable name that represents the aggregation.

## 5.8 File Input/output

File handling was the last stage implemented; it is fairly simple but a critical part of making scribbles a usable application. It has three main sections; diagram file handling, image file handling and java source out put.

The JFileChooser dialogue is used to retrieve information about where a file is to be stored or which file is to be opened. The diagram file handling basically writes all the diagram nodes information and connector information to a text file with extension .sbl. In the main, all elements of a diagram are encodeable as text. The shape type is defined as an integer rather than a GeneralShape [14]. This process is reversed when opening a file, as each element or connector is read from file it is added to a new diagram, once completed this diagram is returned and rendered to the DrawingPanel.

The ability to output a BufferedImage [14] as a JPEG image is inbuilt to Java. This makes outputting images of the class diagram and mind map relatively straightforward.

Outputting Java source code is a similar process to writing the diagram to file. The .java files are just text files with a .java extension. To facilitate minor edits in the editor the actual code saved is retrieved from the editor class rather than generated by the UML model.

# 6 Evaluation

The evaluation of Scribbles looked several areas;

- The effect of Scribbles on early collaborative design
- Value of mind map as an early object model
- Quality of sketch based interaction provided

## 6.1 Group Evaluation

The group evaluation was carried out by a group of six second and third year applied computing students. The session ran for around an hour, and evolved two main problem solving tasks and a five minuet talk about mind maps.

### 6.1.1 Problem One: snakes and Ladders

The initial problem was to design an object model for the game snakes and ladders. The first stage was to mind map the idea and then produce a class diagram. For this task the group worked well together and produced a good class diagram. The group seemed to solve more complex problems at the UML stage and not use the mind map to much. The group also had a single scribe rather than a turn taking approach; this was not prescribed or formally mentioned by any of the group.

From the discussion afterwards the whole team were happy with the solution. They felt they had all had an equal input to the solution. They felt hesitant about committing ideas to paper before they were discussed as the paper diagram can't easily be changed. An implication of this was noted that complex points were discussed before there commitment to paper.

### 6.1.2 Problem Two: Virtual Kitchen

The second problem was to model a program that would act as a virtual kitchen, the graphics were not to be worried about just the logic. This problem was solved using scribbles delivered on a tablet PC with a projector. There was a five minute demo of what Scribbles can do and how to do it. All of the team had played with Scribbles in their own time.

It was evident from the start that there was a significant load when developing ideas under scribbles, this was expected as although the team had some experience of Scribbles the process hadn't become completely natural.

Once the team got into the problem this load was less evident, and complex problems were discussed. They adopted the same set up with a single scribe. The system did encourage early thoughts of relationships which were missing from the paper session. From observing the session it was evident that recognition was not optimal there were a recorded 10 unknown shape entries which is not acceptable for a problem of this size. The rest of the sketch based interface was used with ease and precision.

### 6.1.3 Results and conclusions

To take some more formal feedback from the session a web delivered questionnaire was completed by all the participants.

It was clear that the recognition was not up to scratch. All participants noted incorrect shape recognition rates of 7-9 and 10 plus. Clearly for this to be productive this would need to be reduced. The other clear feedback was more encouraging, all participants preferred scribbles to paper. The main reason being the diagram can be edited easily which is not possible on paper. The imposition of structure was also noted as a good thing.

The primary concern of this evaluation was the impact on team cohesion and productivity. Not one of the participants noticed any difference in their level of their involvement between the two sessions. The majority of the team gave scribbles an obstruction rating of 1 on a scale of 0 to 3. This is not bad considering the relatively limited exposure to the software.

## 6.2 Individual Evaluation

The individual evaluation was designed to explore the sketch interface and mind map model concept in more detail. Participants were walked through scribbles and then allowed to play for around ten minutes. Again a web delivered questionnaire was completed by all the participants.

This questionnaire focused on quality of the sketch interface, concept of mind map, and UML editing. There were mixed result on the quality of the sketch interface I

feel this can be attributed to the graphics tablet used as I was unable to obtain a tablet PC for this evaluation. The graphics table does take some time to become accustomed to this caused some evaluators problems.

The mind map concept was met with great acceptance. All participants found it very useful. The average score (0-5) was 4.3, which was very encouraging.

The UML editing was mastered by most evaluators. The main problem here was the static nature of the UML, having to move things around in diagram view to stop UML classes overlapping was causing a great deal of irritation. The properties panel was liked in comparison to Rational Rose's non modal dialogues. The delivery of the relationship was also praised for its clarity and ease of use. For those who looked at the code Generation they felt the code was well structured and accurate.

## 6.3 Resultant development

Initial I had not planned to have a post evaluation development stage, after the feedback from the evaluation I felt it was worth fixing the small problems in an otherwise well accepted tool.

The recogniser has a two parameters that can be altered easily, the threshold for a non recognised shape and the proportionate thickness of the generated template. The proportionate thickness was set to a tenth of the shortest length (height or width). This factor has been changed to a fifth, this has the effect of making the template thickness grow more rapidly in proportion to the shape size. This has decreased miss classifications.

The ability to move UML classes around has also been added. This was approached with caution as the UML class position is based on the position of the class in the diagram, this is to make the UML layout the same as the diagram. Any changes on the class position in UML view would reflect in Diagram view. If only the class is moved this can have adverse affect to the diagram. To provide this functionality without this problem it was decided to move the classes method, and variable (children) by the same amount. The class's methods and variable are generally in a local cluster around the given class. Moving a class and children produces satisfactory results.

Another option would be to maintain a UML view position and a Diagram view position. This would remove the impact of moving something in one view impacting another view. I didn't take this approach as it starts to erode the association between the UML view and the diagram view. It would be interesting to re run the evaluations with these changes in place. Sadly time will not allow.

## 7 Testing

As this project has been based on week long iterations of design, implement and review I see little need for high coverage testing. Each unit of the problem has been implemented to a satisfactory degree before the next unit is started on. Thus spending time on formally testing each unit is not valuable at this point. These reviews are documented in the review section of the meeting minutes.

To ensure the system forfills the requirements stated a series of acceptance tests have been devised. Acceptance testing is essentially the process of trying to execute the use cases [appendix A](#) in the final system.

In a real world situation this style of testing is the last stage and ultimately finalizes the contract between developer and client; if the tests are successful that is.

All testing carried out is documented in the Acceptance test Plan [appendix D](#)

## 8 Summary and Conclusions

Scribbles has been a valuable opportunity to use all the skills gained in the Applied Computing degree programme and build upon them. This project required a broad spectrum of techniques including; computer vision algorithms, complex data structures, usable interface design, cutting edge software development methodologies, mind mapping and 'pen and paper' interface paradigms. Taking inspiration from the shortcomings of market leading CASE tools, Scribbles has grown to be an advanced prototype of what could be a very powerful tool.

## 9 Future Plans

The scribbles project leaves many interesting possible areas for development;

### 9.1 Short term plans

#### 9.1.1 Cut, Copy and paste

The functionality to cut, copy and paste would be extremely useful. This would be implemented by maintaining a 'clipboard'. This would probably be a stack of diagrams to allow several levels of cutting and pasting. When a copy event was fired a deep copy of the selected items would be added to a new diagram this would be pushed onto the top of this clipboard stack. If a cut event was fired once the copy had occurred a delete event would be fired.

When a paste event was fired the top diagram from the clipboard stack would be popped. This would then be merged the active diagram. Again deep copies would take place this is to make sure the node and connector identification numbers are not duplicated.

This style of multiple levels of *copy* and *cut* would probably need to be supported by an interface component delivered on the properties panel. A visualisation of the copy and the ability to *paste* it by clicking on it would be sufficient.

#### 9.1.2 Undo

The ability to undo the past few actions is a feature that all applications offer nowadays. To support this; a stack of Diagram objects would be maintained. This size of the stack would dictate the number of consecutive undo events possible. As diagram objects are not particularly memory hungry this could be a large number. Every time a diagram altering event is fired the current diagram would be pushed on to the stack.

When an undo even is triggered the stack would be popped and the popped diagram object rendered to the drawing panel.

#### 9.1.3 Requirements gathering

The concept of mind map is a powerful one, in the requirements of scribbles a mind map is used to document all the requirements. The future aim would be to add the ability in scribbles to add text to the nodes. They could be thought of as notes. In the case of requirements gathering the mind map structure would describe the hierarchy of the requirements. The ability to auto generate a requirements document would be possible. It would be even more powerful it could generate Microsoft Word documents.

## 9.2 Long term plans

### 9.2.1 Full synchronisation

To make the process of object modelling programmes really valuable the ability to work from model to code and from code to model is required. This is know as forward and reverse engineering. Scribbles supports forward engineering which is some time referred to as model driven architecture.

The first level of reverse engineering would involve code being edited in Scribbles the ability to edit 'generated' code would be suppressed. When code is entered in to method skeletons and constructors it would be attached to the associated diagram element. This would force editing to be done on the model. This may or may not be a good thing, in the first case it would seem hugely obstructive but it may lead to better code and more consideration about the architecture throughout development.

For full reveres engineering It would be necessary to parse a source code file and generate a model. This is an interesting problem. It would be possible to implement a Ad Hock parser based on some observed rules. For a more robust and re usable solution the use of 'Backus-Naur Form' generative grammars would be wise. This would allow a general solution which could be easily tailored to different languages.

### 9.2.2 Different languages

As is mentioned the code generation is performed in a self contained fashion. It would seem logical to implement code generators for different languages.

### 9.2.3 Different models

The concept of sketch based modelling is a powerful one. It would be possible to hack scribbles to support different model such as Entity relationship models. An elegant solution to providing similar functionality would be to have a diagram grammar. This would describe node types and legal interactions.

### 9.2.4 Export class diagram as XMI

There are a vast number of CASE tools on the market each with there own file format and each with there strengths and weaknesses. This poses a problem for the Software development team a means to transfer models between these tools is offered by the Object Management Group in XMI [15]. XMI is a sub XML based format for the description of UML. The ability to support XMI as an input format and to give the ability to export as XMI would be a useful future development.

XMI was looked at as a primary file type but was declined on the basis that the current version doesn't support some important diagram information such as positions [16].

### 9.2.5 Gesture recognition

The shape recognition algorithm used in scribbles is fairly simplistic; Occam's Razor states that: the simplest solution is generally the best. In this case scribbles is capable of classifying a scribble as a ellipse, rectangle or non shape. The fact a simple algorithm is used is not a problem as it meets the requirements set. This algorithm would probably scale for a larger collection of simple geometric forms or other templates with no fine detail. Therefore building in support for flow charts would probably be possible.

To extend the interface much beyond this would undoubtedly need a more complex recognition process. The first alteration I would make would be to support Gesture recognition. This would store information about the scribbles direction and velocity at a time t. This would provide a vector for each scribble which describes the position, direction and velocity at a given time. A classifier could then consider the changes in direction and velocity rather than just the pixel arrangements. This system would have the advantage of being trainable also. This is a desirable enhancement and could easily consume as much time as the whole project has so far.

## Acknowledgments

This project has been assisted by many people:

**Dr Glenn Rowe** has provided guidance and support all the way, and has focused my enthusiasm throughout.

**Craig Ramsay** provided an early glimpse into finer granularity modelling tools and also a into object modelling in general.

**Dr Anna Dickinson** gave guidance in relation to the evaluation process.

**Raft project** including Dr Nick Hine and, Ms Rosaleen Rentoul for making tablet PC available often at short notice.

**Evaluators:** including second and third year students and members of staff.

## References

- [1] M. Fowler and K. Scott, "UML Distilled: A Brief Guide to the Standard Object Modelling Language (3rd Edition)", Addison Wesley.
- [2] Unified Modelling Language Resource page, <http://www.uml.org>
- [3] T. Buzan, "Use Both Sides of your Brain", Plenum, 1989.
- [4] Buzan Mind Mapping, <http://www.mind-map.com/EN/mindmaps/definition.html>
- [5] A. R Ehrlich, "Mind Mapping an overview and Biography", [http://studenttabletpc.blogspot.com/the\\_student\\_tablet\\_pc/files/mind\\_mapping\\_overview.pdf](http://studenttabletpc.blogspot.com/the_student_tablet_pc/files/mind_mapping_overview.pdf)
- [6] Farrand, Paul, Hussain, Fearzaba, Hennessy, Enid, "The efficacy of the 'mind map' study technique", Medical Education , vol. 36, pp. 426-431, May 2002.
- [7] Mento, Anthony J., Jones, Raymond M., "Mind mapping in Executive education: applications and outcomes", Journal of Management Development, vol 18, April 1999.
- [8] C. Alvaradi and R Davis, "Preserving the freedom of Paper in a computer Based Sketch Tool", Human Computer Interaction International Proceedings, 2001
- [9] Damm, C.H., Hansen, K.M., Thomsen, M., Tyrsted, M., "Creative Object-Oriented Modelling: Support for Creativity, Flexibility, and Collaboration in CASE Tools", Proceedings of ECOOP'2000. Sophia Antipolis and Cannes, France, June 12-16.
- [10] Ideogramic, <http://www.ideogramic.com/>
- [11] Extrem Programming: a Gentle introduction, <http://www.extremeprogramming.org/>

- [12] scrum??
- [13] template matching
- [14] Java 1.5 API,  
<http://java.sun.com/j2se/1.5.0/docs/api/>
- [15] XMI\_specification,  
<http://www.omg.org/technology/documents/formal/xmi.htm>
- [16] C. H. Damm, K. M. Hansen, M. Thomsen and M. Tyrsted, "Tool integration: Experiences and Issues in Using XMI and Component Technology", Proceedings of TOOLS Europe. Mont St Michel & St Malo, France, June 2000