In Proceedings of the 2007 International Conference on Software Engineering, Minneapolis, USA, 21-26 May 2007, IEEE.

# Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool

John Grundy[1, 2] and John Hosking[2]
[1]Department of Electrical and Computer Engineering and [2]Department of Computer Science
University of Auckland, Private Bag 92019, Auckland 1142, New Zealand
{john-g, john}@cs.auckland.ac.nz

## Abstract

*Software engineers often use hand-drawn diagrams as preliminary design artefacts and as annotations during reviews. We describe the addition of sketching support to a domain-specific visual language meta-tool enabling a wide range of diagram-based design tools to leverage this human-centric interaction support. Our approach allows visual design tools generated from high-level specifications to incorporate a range of sketching-based functionality including both eager and lazy recognition, moving from sketch to formalized content and back, and using sketches for secondary annotation and collaborative design review. We illustrate the use of our sketching extension for an example domain-specific visual design tool and describe the architecture and implementation of the extension as a plug-in for our Eclipse-based meta-tool.*

## 1. Introduction

Hand-drawn sketches are often used in software engineering across many phases in the software development process. These include high-level requirements capture, system design, user interface design and code review [2,26,29]. A variety of increasingly popular hardware devices support sketch-based input to computer applications, including the Tablet PC, mobile PDAs, large-screen E-whiteboards, and plug-in tablets for conventional PCs and laptops. Much recent research in HCI and user interfaces has demonstrated the potential of such sketching-based user interfaces to enable more human-centric interaction with computers and to enhance the efficiency and effectiveness of user interfaces, particularly for early-phase design and during collaborative work [5,8,11,15,16,29].

However, most existing software engineering tools lack support for sketching-based input, with the exception of informal annotation in a few tools, e.g. [7]. A small number of design-oriented applications have attempted to provide sketching-based UML and user interface design support [5,8,9,20,28], and a few applications have leveraged sketching-based input for code review and to facilitate communication for collaborative work support [8,16,30]. However all of these systems use either special-purpose tool implementations with limited functionality and integration support or ad-hoc techniques to add sketching-support into existing tools and provide very limited user control over the recognition and formalization of sketched content.

We have developed a meta-tool for building a wide range of domain-specific visual language tools for software engineering design tool development and other diagrammatic modelling applications [14,33]. We have also developed stand-alone, ad-hoc sketching support for early-phase UML design [5]. The success of the latter suggested the usefulness of adding sketching features into our design tool meta-toolset. This would allow any diagram-centric design tool generated by the meta-tool to provide flexible sketching-based input. Given the generality of the meta-toolset, we wanted to provide users with flexible control over the approaches used for sketched content input and processing. To achieve this, we have enhanced our meta-tool, which is realised as a set of Eclipse plug-ins, with an extra plug-in to support flexible sketch-based input in any generated tool implementation. The support provided includes both eager and lazy shape recognition; progressive formalization of sketches into computer-drawn content; preservation of sketched content; and the ability for users to easily move between sketched diagrams and formalized diagram content, or even to mix the two. Our generated design tools with sketch-based input run as Eclipse plug-ins. While our sketching support currently only works for our Marama-implemented tools the use of Eclipse does allow close integration with other Eclipse-based software engineering tools.

We first introduce a motivation for this research and identify a set of key requirements for generic software tool sketch-based input support. We survey related research in this area and outline the main features of our approach, providing an example illustrating the use of our sketching-based design tools. The architecture and implementation of our meta-tool and the additional sketching plug-ins are described. We finish with discussion of our experiences with these tools and their strengths and weaknesses and key areas for future research.

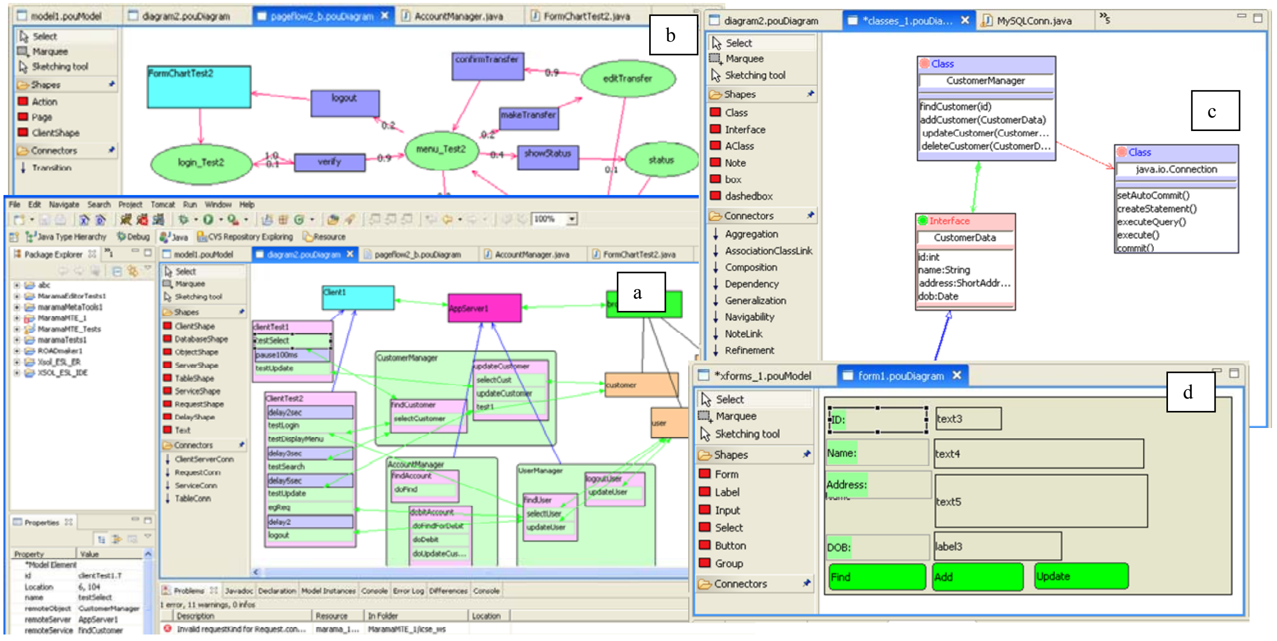


**Figure 1. Examples of software design with MaramaMTE.**

## 2. Motivation

Consider the design of a complex software architecture. Engineers often use multiple views to capture key architectural requirements, key architectural abstractions, and structural vs. behavioural aspects of architectural components [18]. In doing so, they often sketch out preliminary designs for the system architecture, refining the views as design progresses. They may review their designs collaboratively with other engineers and developers before moving into detailed design and implementation of the architecture using a variety of design and coding tools.

We have developed several tools to support such architectural capture. One of them, MaramaMTE [10,13], is shown during use in Figure 1. Figure 1 (a) is a structural diagram showing client, server, database and other key structural abstractions. Figure 1 (b) shows a model of client behaviour as pages, actions and inter-relationships. Such an architecture may be refined to or reverse engineered from a set of more detailed UML design diagrams, e.g. Figure 1 (c), and user interface designs, e.g. Figure 1 (d).

During development of MaramaMTE-like design diagrams, and related UML and user interface designs, software engineers may want to use sketches to assist their design conceptualisation. Using hand-drawn designs in this way has the demonstrated advantages over conventional software diagramming tools of flexibility, encouragement of exploratory design, and support for collaborative annotation and design review [2,26,29]. Conventional diagramming tools have been shown by many studies to suffer from premature commitment to particular design artefacts and choices; over-constraint of user actions; high viscosity (making designs hard to change); stifling of creativity; and limited collaborative design and review support [29]. As many studies have shown, sketching-based design tools for a wide range of diagram-centric tasks offer ways of combining the advantages of paper-based and whiteboard-based design with those of computer-based diagramming tools [2,5,7,11,15,20,29]. To date, though, most such research has focused on supporting sketching-based input in a narrow range of design tasks i.e. only providing for very limited diagram types, and in ad-hoc ways i.e. strongly tied to one tool with very limited reusability of sketching and recognition support.

To more widely realise the advantages of sketch input we were motivated to add effective sketching support to all diagram-based design tools realised using our Marama meta-toolset. Marama supports the specification and generation of Eclipse-based design tools, including the MaramaMTE, UML and XForm design tools in Figure 1. Marama tool specifications, designed using a set of primarily visual meta-tools, are loaded by a set of Eclipse plug-ins which generate each diagramming tool. In order to add sketch-based design support for Marama tools we identified the following key requirements:

- *Sketching for any Marama tool.* Sketch-based input, recognition and annotation for any Marama-generated diagramming tool must be supported. Users should be able to “draw”, using a mouse, tablet PC stylus, external tablet or Mimio-style E-whiteboard pen, content that is captured by the Marama design tool. The sketched diagram elements should then be recognised and converted into Marama diagram elements. There should be minimal (or ideally no) modification or extension of the Marama diagramming tool specifications to support sketching-based input.

- *Flexible recognition and conversion.* Diagramming tool users need flexible control over when content is recognised and converted i.e. eager vs lazy sketched shape recognition. Some diagramming tasks (and user preferences) suit conversion of a sketched shape into a computer-drawn Marama shape immediately it is drawn. Many others better-suit sketching a whole diagram and recognising and converting it as a whole. Still others suit a mixture of approaches, particularly annotation of a conventionally-edited diagram for design review tasks.
- *Recognition accuracy*. Accurate shape and text recognition has been shown to be essential for sketch interfaces [22]. However, users should be provided with the ability to very easily over-ride the recogniser when it makes an error. The recogniser may need to support training to individual user sketching styles [17, 27, 31].
- *Seamless movement*. Users need to be able to move easily between sketching-based diagram input and annotation and conventional mouse-driven editing of diagram content within the IDE.
- *Collaboration support*. Collaborative design and review support should include distributed, multi-user sketch-based diagram input and annotation.

## 3. Related Work

Many CASE tools support UML modelling, almost all using conventional mouse/keyboard input and formalised icons [32]. Industry adoption of these tools has been mixed [2,19] with empirical studies showing designers find them to be overly restrictive during early design with developers preferring to sketch early designs by hand [2,8,11,19,27]. Diagram editing constraints can also be very distracting to users, especially during creative design work [3,19].

There has been considerable work in the area of pen based sketch input of software designs, with support for formalization of sketches into design artifacts. One of the earliest, SILK [20], allows software designers to sketch an interface using an electronic pad and stylus. SILK recognizes widgets and other interface elements as soon as they are drawn and can transform sketches into standard Motif widgets. Denim [21] is a similar approach to SILK but for web interface design.

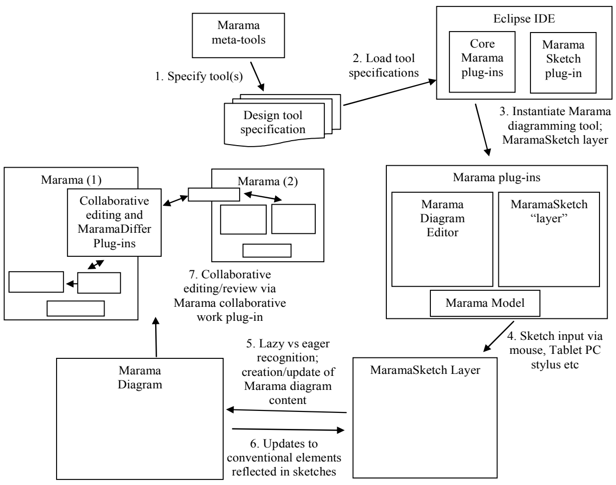
Freeform [28] provides sketch definition and testing of Visual Basic forms. Freeform user studies shows that providing interaction capability with retained sketches encourages more complete exploration of design alternatives. Forms3, a spreadsheet style end user programming environment, has been extended with gestural input [4]. Amulet supports gesture-based document manipulation [25] while Knight [8], SUMLOW [5], and Donaldson et al [9] support UML diagram sketching. Most immediately convert sketched input into computer-drawn formalized content. However, user evaluations for SUMLOW showed that keeping sketched designs is very effective during early phase UML diagramming and when collaboratively reviewing and revising designs with an E-whiteboard. PenMarked provides pen-based code annotation support [30]. Its user studies showed good efficacy of retained pen annotations for code review. These various systems and user studies have affirmed to us that preserving sketch content and having it formalized in flexible ways is both appropriate and useful to support effective software design and review.

Each of these systems is, however, closely tied to the underlying tool it is providing sketch recognition for with little attention to reuse for other sketch-based applications. A variety of low level sketch support tools have been designed with reuse in mind. These include Rubine's [31] single stroke gesture recognition algorithm (used by SILK and Freeform) and Apte's [1] multi-stroke algorithms. Hse has developed the multi-stroke recognition approach into HHReco, a reusable Java toolkit supporting sketching which incorporates a range of trainable and customizable recognisers [17]. While these toolkits are all immensely useful, they still require significant programming to incorporate into other applications. Our interest in this work was in making such generic sketch support available to a wide range of design tools without modification or additional programming on a tool-by-tool basis.

## 4. Our Approach: MaramaSketch

In order to develop a sketching-based extension for Marama diagramming tools we developed a new plug-in, MaramaSketch. This provides an overlay for Marama diagrams allowing sketching-based input and manipulation of diagram content along with associated shape and text recognition support. Figure 2 illustrates the process of using MaramaSketch.

A tool developer uses the Marama meta-tools to specify a design tool (1). A set of core Eclipse plug-ins provides diagram and model management support for Marama modelling tools. A tool user opens or creates a new modelling project and diagrams using these plug-ins. If installed, an additional MaramaSketch plug-in augments Marama diagramming editing with sketch-based input and recognition (2). When a diagram is created or opened in Marama, a sketching "layer" is created and managed by the MaramaSketch plug-in (3). This intercepts mouse/pen input on the diagram canvas when the MaramaSketch input tool is selected by the user. Drawing with the sketch input tool creates sketch layer elements (single-stroke and multi-stroke shapes) (4). Depending on user preferences, sketched input may be: immediately recognised and converted to Marama diagram content; recognised but not immediately converted; or converted on-demand by the user e.g. after a whole design has been sketched (5). The user may select conventional Marama diagram edit tools and modify the Marama diagram content e.g. move, resize or delete Marama diagram elements. Such edits are propagated back to the sketch elements associated with the Marama diagram elements (6). Collaborative editing and review are supported using a further plug-in component.



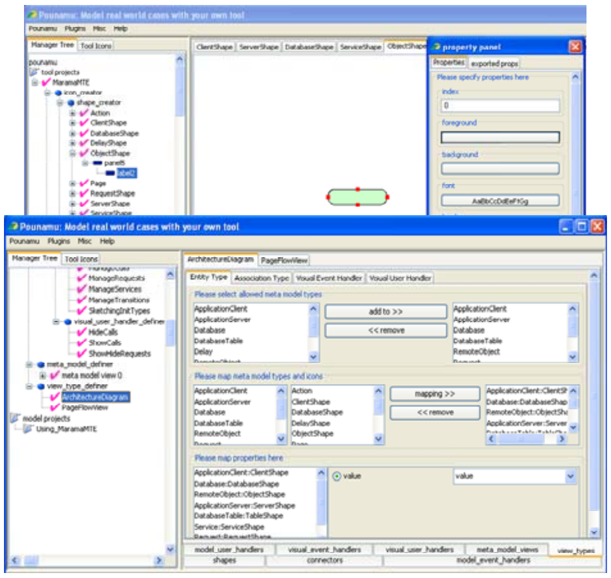
Marama meta-tools
1. Specify tool(s)
Design tool specification
2. Load tool specifications
Eclipse IDE
Core Marama plug-ins
Marama Sketch plug-in
3. Instantiate Marama diagramming tool; MaramaSketch layer
Marama (1)
Collaborative editing and MaramaDiffer Plug-ins
Marama (2)
7. Collaborative editing/review via Marama collaborative work plug-in
Marama plug-ins
Marama Diagram Editor
MaramaSketch "layer"
Marama Model
4. Sketch input via mouse, Tablet PC stylus etc
5. Lazy vs eager recognition; creation/update of Marama diagram content
Marama Diagram
MaramaSketch Layer
6. Updates to conventional elements reflected in sketches

**Figure 2. Using MaramaSketch.**

## 5. Example Usage

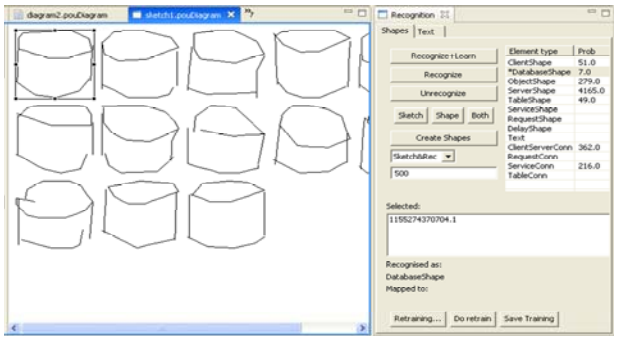
We illustrate MaramaSketch's capabilities via its use with the MaramaMTE architecture design tool shown in Figure 1. Initially a tool developer specifies a diagram-based design tool using a set of visual meta-tools [14]. Figure 3 shows part of the meta-tool definition for the MaramaMTE architecture design tool. Shown is a shape specification (a) - this one is for an *ApplicationServer*, and (b) part of the view type specification (set of shapes and connectors and their relationship to an underlying model) for the *ArchitectureView* diagrams. This Pounamu tool specification is loaded by Marama when requested by the Marama tool user. It provides the available diagram elements that can be input by a tool user and thus that may need recognition from sketched input.

A crucial aspect of the success of sketching-based design tools is accuracy of the shape and text recogniser(s) employed [22]. We chose to use a multi-stroke, training-based shape and text recognition algorithm [17] for MaramaSketch. This was primarily to allow individual users to describe their own examples of each available shape type for MaramaSketch to increase the accuracy of its recognition. Earlier work that we did with a non-trainable recogniser for UML diagramming had insufficient accuracy which became frustrating to users [5]. In addition, as MaramaSketch is intended to support any kind of Marama diagramming tool the available shape types are virtually infinite, leading to eventual difficulty distinguishing between both simple and complex shapes if a non-tool-specific approach is taken. We decided to provide users with the ability to incrementally re-train their MaramaSketch shape and text recognisers while the tool is in use. When a sketched item is incorrectly recognised the user can over-ride the MaramaSketch-recognised shape and ask for the new shape to be added to the recogniser training set. Users can share their training sets so one user might initially specify available shape examples and other may use these, re-training the recogniser over time.



Pounamu: Model real world cases with your own tool
property panel
Pounamu: Model real world cases with your own tool
ArchitectureDiagram
PageFlowView
add to >>
<< remove
mapping >>
<< remove

**Figure 3. Pounamu tool definition examples.**



500

| Element type | Prob |
|---|---|
| ClientShape | 51.0 |
| *DatabaseShape | 7.0 |
| ObjectShape | 279.0 |
| ServerShape | 4165.0 |
| TableShape | 49.0 |
| ServiceShape | |
| RequestShape | |
| DelayShape | |
| Text | |
| ClientServerConn | 362.0 |
| RequestConn | |
| ServiceConn | 216.0 |
| TableConn | |

Selected:
1155274370704.1

Recognised as:
DatabaseShape
Mapped to:

**Figure 4. Shape recogniser training example.**

Figure 4 shows a user training the MaramaSketch shape recogniser by specifying multiple, multi-stroke examples of a shape. We use various heuristics to identify mouse or stylus strokes as belonging to the same shape, including proximity, time between stroke end/start, and information returned by the recogniser. We use the same algorithm but different training set and stroke grouping heuristics for text recognition.

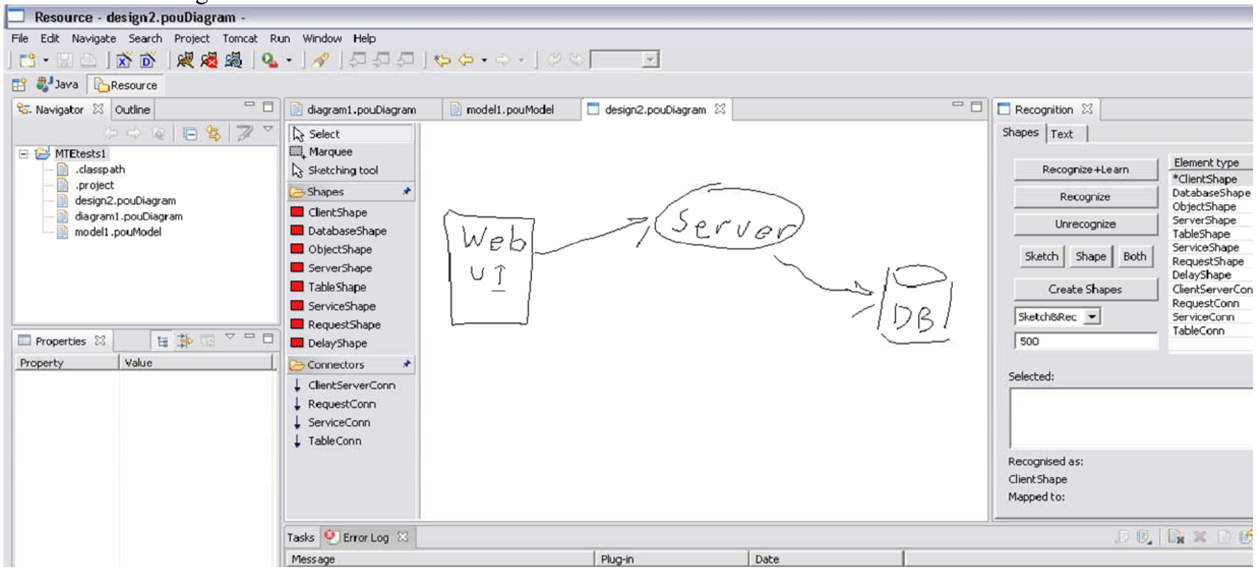


**Figure 5. Drawing an architecture design with MaramSketch.**

Figure 5 shows an example of a user drawing content (in this example with a Tablet PC stylus) onto a MaramaMTE *ArchitectureView* diagram. The user simply selects the sketching tool (highlighted in the left hand side editing palette) and draws with the mouse/stylus on the diagram canvas. In this example the user has drawn a *ClientShape* (rectangle, "Web UI"), an *ApplicationServer Shape* (oval, "Server"), a *DatabaseShape* (cylinder, "DB") and two connections between shapes. As each set of strokes is completed MaramaSketch recognises the shape type and remembers this.

The Recogniser view on the right is normally hidden but as illustrated in this example it shows the MaramaSketch shape recogniser probabilities for the most recently drawn or selected and grouped strokes (in this example a new *ClientShape* sketch). The user may over-ride the recognition and learn the new sketched shape as an example of specified shape type via either a pop-up menu or this Recogniser view. Depending on user preferences the drawn strokes can be (1) left unrecognised; (2) recognised as a Marama shape type or text string; (3) recognised and a Marama diagram shape, connector or text property value created; or (4) recognised and immediately replaced by a computer-rendered Marama diagram shape, connector or text property value. Figure 6 shows examples of each of these approaches.

In (1), the user simply draws multi-stroke shapes and Marama doesn't attempt any recognition or grouping. In (2), the user has asked Marama to recognise and if necessary group strokes. Here, the sketched shape (made up of 4 lines) has been recognised as a *ClientShape* type and the 4 lines grouped into one composite sketched shape. In (3), the sketched shape has been recognised and a Marama *ClientShape* created and its size and location inferred from the sketched stokes. The user has asked that the sketched strokes and new Marama shapes be shown together (one can be switched off by user preference). In (4), a set of strokes making up a *ClientShape* and the client *name* property have been drawn, recognised by MaramaSketch, and converted into a *ClientShape* with *name* set to "Client1". The user has asked for the sketched strokes to be hidden immediately after recognition.

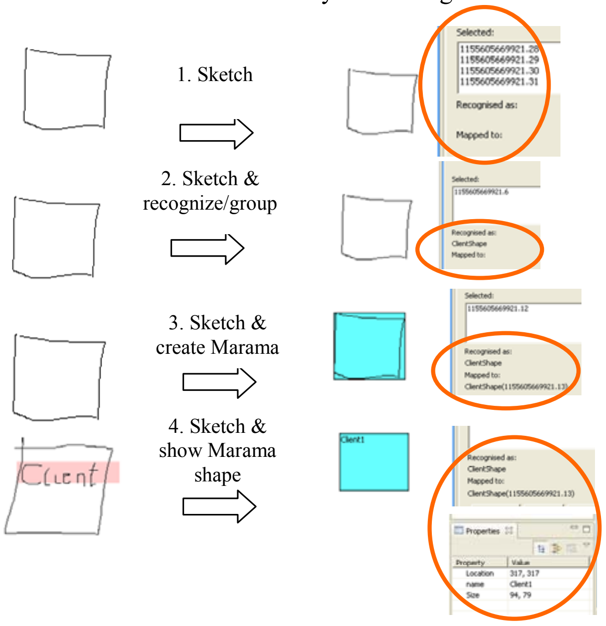


**Figure 6. Recognition approaches in MaramaSketch.**

A key problem in many sketching-based tools is distinguishing between shapes and text (characters and words). These suit different recognition algorithms and multi-stroke grouping heuristics. We chose to delineate between the two in MaramaSketch by use of a technique we developed for our previous ad-hoc UML sketching tool, SUMLOW [5] and is illustrated in Figure 7. As soon as a sketched shape is recognised as a Marama shape or connector one or more explicit “text area” annotations are automatically added to the sketched shape or connector. These “text areas” (rendered as light pink rectangles) have mouse-over tool-tips indicating the shape or connector property to which the text area corresponds. Any sketched content predominantly inside a text area annotation is assumed to be text and is processed using a different recognition algorithm and stroke grouping heuristics. Again the user can override the recognised text using the Recogniser view or by editing the generated Marama shape property value in the Eclipse Properties view. Text areas can be set to auto-hide after text is recognised, reducing diagram clutter. They can be re-shown for a shape by right-click menu option e.g. to allow over-write modification and then re-recognition of the text.

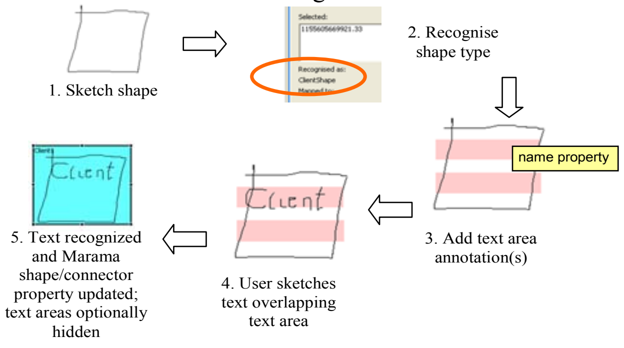

1. Sketch shape
Selected:
1155605669921.33
Recognised as:
ClientShape
Mapped to:
2. Recognise shape type
name property
3. Add text area annotation(s)
4. User sketches text overlapping text area
5. Text recognized and Marama shape/connector property updated; text areas optionally hidden

**Figure 7. Text recognition in MaramaSketch.**

This approach does introduce some premature commitment [12] as shapes must be drawn and recognised before the text annotations can be added otherwise the text won’t be recognised as being associated with shape. In practice, using the approach (2) of Figure 6, this does not prove intrusive to the sketching process as shapes are recognised quickly enough for the annotation areas to be added immediately the shapes are sketched. This still means that shapes must be drawn before text, but this is a fairly natural ordering when sketching iconic shapes so we deemed this limited premature commitment to be appropriate. The user can however force a set of strokes to be recognised as text rather than as a shape or connector by using a right-click menu option.

A similar ordering constraint is currently used when recognising connectors i.e. lines (possibly with arrows and/or other annotations) between shapes. MaramaSketch firstly recognises the source and target shape types and uses these to inform the recogniser of likely connector type from the Marama meta-model for a diagram. This often greatly reduces the possible connector types possible. For example, MaramaMTE client and server shapes can only be linked by a “ClientServerConn” connector type, hence any connector drawn between them must be of this type.

Users can switch between the different recogniser approaches as sketches are drawn allowing a mixture of sketch and formalised diagram elements to appear in the one diagram. Figure 8 (left) shows an example of this. The initial diagram drawn in Figure 5 has been recognised using Approach (3) with formalized Marama shapes and connectors overlaid by the original sketched shapes. An additional *ServiceShape* (rounded rectangle, “Customer Service”) with an embedded *ObjectShape* (rectangle, “Cust”) and two connectors have been added using Approach (2) which leaves the new shapes and connectors in sketched form only. The diagram may be fully formalized at any time. The user may also add a formalised shape directly via the tool palette and one of the shape training examples is added to the sketch layer to represent this (currently just the first training example).

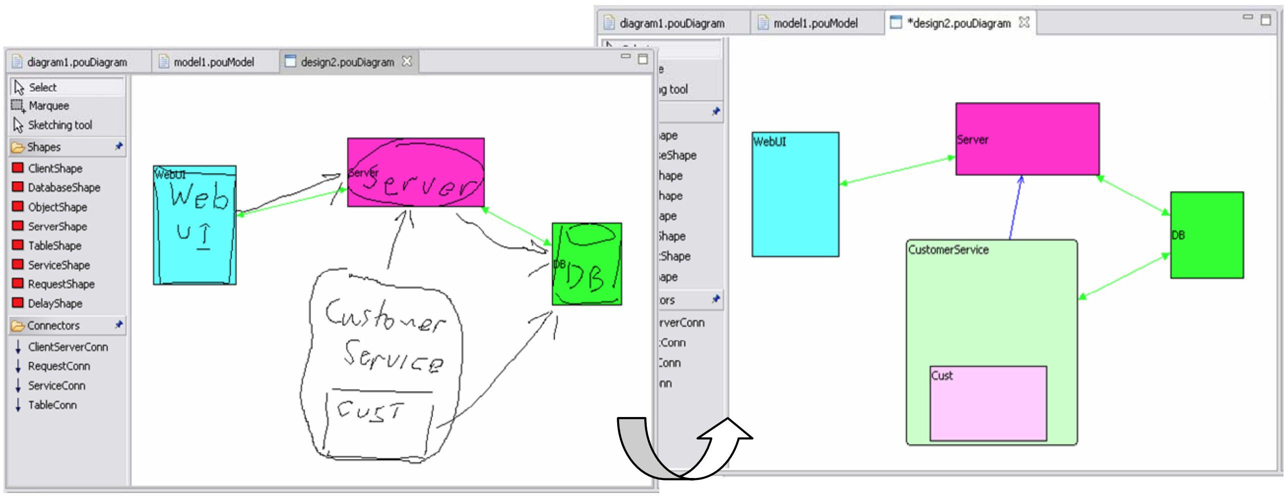


**Figure 8. Mixing sketches and Marama diagram elements.**

The Cust sketch is recognised as an *ObjectShape* due to its placement inside the ServiceShape; only remote objects in MaramaMTE can be placed here. MaramaSketch uses this syntactic information from the Marama diagram meta-model to reduce the possible match options for the shape recogniser and hence improve recognition rates. The fully recognised architecture diagram is shown to the right. The user may freely alternate between the formalised and sketched representations.

Diagrams can have secondary notation added, as is shown in Figure 9. Here a preliminary design is being critiqued with sketched annotations added to capture elements of the design review. These are not recognisable as Architecture Diagram shape types so are ignored by the Recogniser but retained as secondary notation. The user can explicitly stop recognition if desired when doing such “informal annotation” of a diagram.

These annotations also provide an effective collaborative review mechanism where users share these via synchronous or asynchronous editing support. MaramaSketch supports sharing of sketched content via a set of synchronous editing plug-ins we developed for standard Marama diagram synchronous editing [23]. Asynchronous sharing is supported by a shared CVS repository and diagram diffing and merging support, also from Marama plug-ins [24].

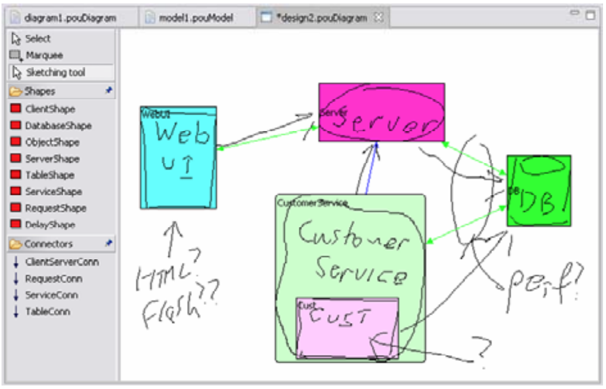
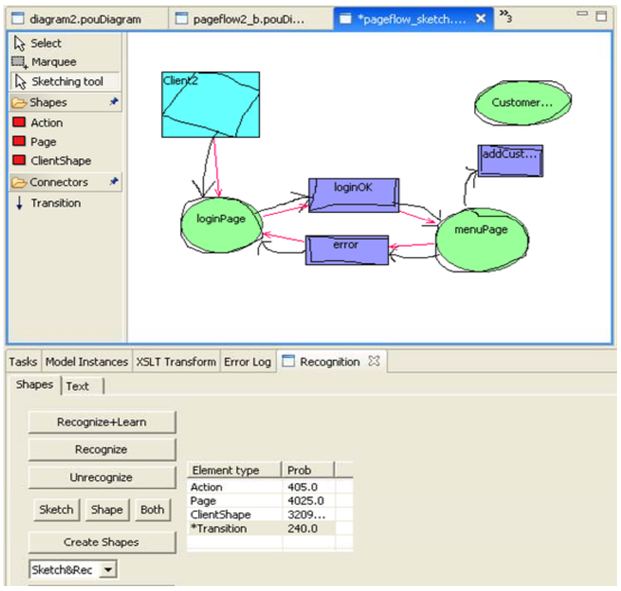


**Figure 9. Annotating and reviewing designs.**

Figure 10 shows another sketched diagram for a different view type, this time a page flow view. MaramaSketch uses different training sets for each shape/connector type. When a view type is defined as being composed of a particular set of shape and connector types, the matching set of training sets is used by MaramaSketch to recognise shapes in that view type. Thus, although some of the shapes in Figure 10 are similar to those in Figure 9, they are appropriately recognised as page flow elements.

## 6. Design and Implementation

We developed MaramaSketch on top of our Marama Eclipse-based diagramming toolset [14]. Marama leverages Eclipse’s EMF and GEF frameworks to provide a wide range of diagram editing tools. MaramaSketch extends Marama to provide a sketching layer on top of conventional Marama diagramming tools. The high-level design of MaramaSketch is shown in Figure 11.



| Element type | Prob |
|---|---|
| Action | 405.0 |
| Page | 4025.0 |
| ClientShape | 3209... |
| *Transition | 240.0 |

**Figure 10. Using MaramaSketch with other view types.**

MaramaSketch adds a set of extra view-level components to a Marama diagram: TimedPoints; SketchedShape; and GroupedSketch. TimedPoints capture X,Y co-ordinate and millisecond timing as the user draws on the Marama diagram canvas. The recorded timing information is used by the shape recogniser. TimedPoints are aggregated into a SketchedShape which represents a single stroke shape. Each of these "strokes" may be further aggregated into a GroupSketch, a set of SketchedShapes. A SketchedShape or GroupedSketch may be recognised as and related to a Marama editor shape, connector (line between shapes) or property value (if the GroupedSketch has been recognised as text). The MaramaSketch components form a "layer" above the conventional Marama diagram editor shapes and may be shown or hidden as illustrated previously. They are saved and loaded to the same XMI-format file as the Marama Shape, Connector and Property components for the diagram they have been added to.

Detailed information about Sketched shapes can be viewed and modified via the MaramaSketch Recogniser view, as illustrated previously. This view allows users to explicitly group, ungroup, over-ride the recogniser add new examples to the recogniser and recognise text or graphical shape content. In addition, this augments the Marama editor that has the sketch layer with a set of pop-up menus allowing the user to non-modally over-ride the recogniser, learn new examples, ungroup recogniser-grouped sketched shapes etc.

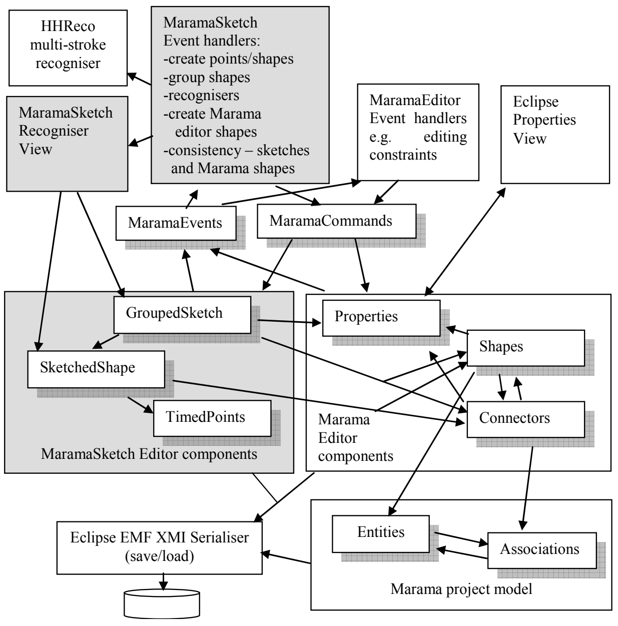

HHReco multi-stroke recogniser
MaramaSketch Event handlers:
-create points/shapes
-group shapes
-recognisers
-create Marama editor shapes
-consistency – sketches and Marama shapes
MaramaSketch Recogniser View
MaramaEditor Event handlers e.g. editing constraints
Eclipse Properties View
MaramaEvents
MaramaCommands
GroupedSketch
SketchedShape
TimedPoints
MaramaSketch Editor components
Properties
Shapes
Connectors
Marama Editor components
Eclipse EMF XMI Serialiser (save/load)
Entities
Associations
Marama project model

**Figure 11. Key architectural abstractions of MaramaSketch.**

In Marama, diagram editing constraints and controls are supported via a set of "event handlers" that subscribe to MaramaEvents and describe changes to Marama diagram shapes and connectors. We implemented a set of event handlers for MaramaSketch that listen to changes made to both MaramaSketch layer components and Marama editor components e.g. move mouse, which create TimedPoints, add SketchedShape, resize SketchedShape, move MaramaShape, delete MaramaConnector etc. These event handlers provide the essential MaramaSketch functionality of creating sketched content; grouping sketched shapes; recognising sketched shapes; creating Marama shapes and connectors and relating them to sketched shapes and groups; and modifying sketched shapes when Marama shapes/connectors have been edited and vice-versa. The event handlers make changes to MaramaSketch components and Marama diagram components by creating and running MaramaCommands on them that effect required state changes.

MaramaSketch uses the open source HHReco toolkit to support multi-stroke text and graphical shape recognition [17]. HHReco provides an incrementally re-trainable set of positive and negative examples that can be augmented incrementally during MaramaSketch usage or via custom training sets developed before use. We use two differently configured HHReco recognisers, one for graphical shape recognition and one for textual character recognition. The graphical shape recogniser uses a set of heuristics to group multiple drawing strokes into shapes for recognition. These include time between strokes, stroke overlap, and recogniser probabilities returned when trying different groupings of multiple shapes in sequence.

The HHReco-based text recogniser supports multi-stroke character recognition, in contrast to the common single-stroke approaches such as Rubine's algorithm [31] and the original Graffiti [22]. Disambiguation as to whether Strokes belong to text vs graphical shapes is currently managed using "text area" annotations, greatly improving recognition rates. Our text recogniser for MaramaSketch could be replaced with e.g. the native text recogniser in the Tablet PC operating system. However this would require running MaramaSketch only on a Tablet PC and would require detailed data structure and API call changes. HHReco allows MaramaSketch to be used on any computing platform with mouse-based input, making the implementation much more portable.

## 7. Discussion

We have so far used the MaramaSketch plug-in to augment a software architecture design environment (MaramaMTE), a web service composition tool (ViTABaL-WS), a simple UML class diagramming tool, and a music composition tool. No code changes were required for MaramaSketch to work for any of these tools –a single extra event handler is added to the tools' meta-tool specification to initialise the MaramaSketch capabilities when a diagram is opened. The plug-in has been used on a conventional desktop PC and on a tablet PC and works on either without modification. Due to the prototypical nature of MaramaSketch, in particular the unintuitive user interface provided by the recogniser view, we have not yet conducted an empirical usability study of the plug-in. Instead we have demonstrated the augmented software architecture and web service composition tool to several experienced users and developers of Marama tools and to two novice users of the music composition tool. We obtained preliminary feedback on its potential usefulness in these domains.

Key strengths of the approach we have taken in MaramaSketch include:

- It is generic, working for any Marama tool – even for the design of Marama tools (as our meta-tools are themselves Marama tools). This is in comparison to approaches such as SUMLOW [5] and Knight [8] which are limited to one toolset only.
- It is highly flexible, in that it can be tailored to suit both the tool and end user preferences in terms of its recognition strategy and also in the sketched symbols it will recognise (as embodied in its training sets). Again, this compares favourably to other sketch tools which limit end user choice [5, 8].
- It provides seamless movement both ways between sketching and formalised diagram manipulation.
- It is highly platform portable, limited only by the portability of the underlying Eclipse toolset that it is based on.

Current weaknesses of our approach include:

- The need for training sets. Although they are a key to the tool's flexibility, they take time to set up when defining a tool. However, this time is amortised over (typically) many applications of that tool definition.
- The user interface is somewhat clumsy when over-riding mis-recognised shapes and the prototype recogniser viewer is unintuitive for most users
- The selection of recognition modes e.g. recognise & automatically create shape by users is unintuitive
- The automatic "divider" that determines when to recognise a set of strokes as shape or text is very rudimentary and prone to error
- There is some premature commitment in the approaches taken for text annotation and for connector differentiation, as discussed in the previous section.
- The Marama meta-tool specifications currently have limited information about complex shape relationships e.g. containment and alignment, which if improved would assist shape recognition by reducing options
- It only works for diagramming tools developed using our Marama meta-toolset

The key requirements expressed in Section 2 have all been met. Genericity, flexibility, and seamless movement are described above as key advantages. Recognition accuracy is high and the incremental nature of the training sets means that accuracy can be improved for individual users over time to suit end-user symbol specification preferences.

Premature commitment, which we have discussed in some detail, is one of many dimensions in the Cognitive Dimensions of Notations Framework (CD) [12]. We have used CD to assist us in the design of MaramaSketch. Dimensions we have emphasised, in addition to premature commitment include:

- *Viscosity*: pen and paper/whiteboard sketching has high viscosity; i.e. it takes considerable effort to change a diagram element. In MaramaSketch we have attained much lower viscosity by permitting sketched elements to be resized/moved using the mouse or pen.
- *Progressive evaluation*: we have aimed for high progressive evaluation. End users can have their sketches recognized at any time allowing them to obtain feedback as and when they desire on whether their sketches have been recognised correctly (and can override that recognition if they haven't)
- *Secondary notation*: again, we have aimed at high secondary notation support. End users can selectively turn off recognition to add any desired form of secondary annotation (or may simply annotate using symbols that are not recognised). This allows arbitrary secondary annotation to be added to any diagram.

- *Closeness of mapping*: this dimension was central to our motivation i.e. that sketching is a more natural mechanism for expressing initial designs than standard computer diagramming approaches.
- *Error-proneness*: The tool currently delineates shapes from text with user assistance (dynamic text areas on shapes) and simple heuristics. While this works if used the way we intended, this approach introduces *premature commitment* and fails if text and shapes are attempted to be recognised in one batch operation.

Other dimensions were less relevant to MaramaSketch's design, as they are more specific to a particular notation/tool implemented by Marama rather than the generic support of MaramaSketch.

There are several areas of improvement that could be made to MaramaSketch. The current implementation uses the HHReco toolkit for text recognition for reasons of portability. Supplementing this with platform specific recognition capability where this is available, such as the Tablet PC text recogniser, would greatly improve recognition performance – particularly for text – at the expense of having to maintain multiple architectures.

An alternative approach to using the text area method for text annotation delineation would be to use a "divider" algorithm, such as is used in the Tablet PC, to automatically infer the distinction between text and graphical objects prior to detailed recognition. This would eliminate the premature commitment issues discussed earlier. The Inkkit toolkit [6] could be used for this purpose. Its divider performance is significantly better than that of the Tablet PC, however it is still platform specific and hence would limit portability.

The current system provides a limited form of "deformalisation" of a standard Marama diagram element i.e. "re-engineering" a sketch from the standard Marama shapes and connectors into realistic-looking sketch elements. As discussed earlier, there is ample evidence to suggest that sketched diagrams encourage designers to explore and critique designs more thoroughly so conversion of formal diagrams into sketches could be useful to encourage that process. MaramaSketch currently provides a limited form of this by selecting the first sketched shape from its training sets to replace formal shape and connector elements in a diagram. These are crudely resized and then combined with similarly generated text annotations. Understanding whether the sketches were then sufficiently realistic to encourage the desired behaviour would then need to evaluated empirically.

An additional application that MaramaSketch could be extended to is annotation of Eclipse code views. This would use the same sketch overlay mechanism, but to support code annotation and review rather than diagram construction. This would provide a similar mechanism for Eclipse as Plimmer and Mason [30] have provided for Visual Studio. It may be possible to seamlessly augment any Eclipse GEF (Graphical Editing Framework)-based diagramming tool with a MaramaSketch overlay.

## 8. Summary

We have described MaramaSketch which generically extends tools generated by our Eclipse-based Marama meta-toolset with sketch input capabilities. The sketching extension is tailorable in its recognition approach, spanning the spectrum from lazy through eager recognition and is incrementally trainable to cope with idiosyncrasies of individual users. Experience with this approach has been promising for providing truly generic sketch input support for software engineering diagramming tools.

## References

1. Apte, A. Vo, V. Kimura T. D. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In Proc UIST 1993, ACM Press, pp. 121-128.
2. 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.
3. Brooks, A. and Scott, L. Constraints in CASE Tools: Results from Curiosity Driven Research, In Proc ASWEC 2001, , 26-28 August 2001, IEEE CS Press, pp. 285-296.
4. Burnett, M. and Gottfried, H Graphical Definitions: Expanding Spreadsheet Languages through Direct Manipulation and Gestures, ACM TOCHI 5(1), 1-33, 1998.
5. Chen, Q., Grundy, J.C. and Hosking, J.G. An E-whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams, Proc HCC'03, Auckland, October 2003, 219-226.
6. Chung, R., P. Mirica, and B. Plimmer. *InkKit: A Generic Design Tool for the Tablet PC*. Proc *CHINZ 05*. 2005. Auckland: ACM: p. 29-30.
7. Churcher N, Cerecke, C, groupCRC: Exploring CSCW Support for Software Engineering, Proc OZCHI'96, 62-68
8. Damm, C.H., K.M. Hansen, and M. Thomsen. *Tool support for cooperative object-oriented design: Gesture based modelling on and electronic whiteboard*. Proc *Chi 2000*. 2000: ACM: p. 518-525.
9. Donaldson, A. and Williamson, A. Pen-based Input of UML Activity Diagrams for Business Process Modelling, Proc HCI 2005 Workshop on Improving and Assessing Pen-based Input Techniques, Edinburgh, September 2005.
10. Draheim, D., Grundy, J., Hosking, J., Lutteroth, C., Weber, G., Realistic Load Testing of Web Applications. Proc CSMR 2006, IEEE CS Press, 57-70, 2006
11. Goel, V., Sketches of thought. 1995, Cambridge, Massachusetts: The MIT Press.
12. 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.

13. Grundy, J., Hosking, J, Li, L., Liu, N., Performance Engineering of Service Compositions, Proc IW-SOSE 2006, 26-32, May 2006.
14. Grundy J., Hosking, J, Zhu, N., Liu, N., Generating domain-specific visual language editors from high-level tool specifications, accepted for Proc ASE 2006, to be held in Tokyo, Sept 2006.
15. Hearst, M.A. Sketching intelligent systems, IEEE Intelligent Systems and Their Applications, 13(3), 10-19, 1998
16. A. Huang, T. W. Doeppner, U. C. Readers, "Ad-hoc Collaborative Document Annotation on a Tablet PC", 2003
17. Hse, H. and Newton, A.R. Robust Sketched Symbol Recognition using Zernike Moments, In Proc. ICPR, Aug. 2004, Cambridge, UK.
18. IEEE Recommended practice for architectural description of software-intensive systems, IEEE Std 1471-2000, 2000
19. Iivari, J. Why are CASE tools not used? Communications of the ACM, vol. 39, no. 10, October 1996, 94-103.
20. Landay, J. and B. Myers. *Interactive sketching for the early stages of user interface design*. Proc *Chi '95 Mosaic of Creativity*. 1995. ACM: p. 43-50.
21. Lin, J., Newman, M.W., Hong, J.I. and Landay, J. A. Denim: Finding a tighter fit between tools and practice for web design, Proc CHI'2000, ACM Press, pp. 510-517
22. MacKenzie, I. S., & Zhang, S. The immediate usability of Graffiti. Proceedings of Graphics Interface '97, pp. 129-137.
23. Mehra, A., Grundy, J.C. and Hosking, J.G. Adding group awareness to design tools using a plug-in, web service-based approach, Proc 6th Int Workshop on Collaborative Editing Systems, 2004, http://cocasoft.csdl.tamu.edu/~lidu/iwces6/
24. Mehra, A., Grundy, J.C., Hosking J.G., A generic approach to supporting diagram differencing and merging for collaborative design, *2005 IEEE/ACM ASE,* 204-213
25. Myers, B.A. The Amulet Environment: New Models for Effective User Interface Software Development, *IEEE Trans SE*, vol. 23, no. 6, 347-365, June 1997
26. C. C. Marshall, Annotation: from paper books to the digital library, 1997
27. Plimmer, B. Apperley, M. Computer-aided sketching to capture preliminary design. Proc AUIC 2002, Australian Computer Society, Inc, pp. 9-12.
28. Plimmer, B.E. and M. Apperley. *Software for Students to Sketch Interface Designs*. Proc *Interact*. 2003. p. 73-80.
29. Plimmer, B.E. and M. Apperley. INTERACTING with sketched interface designs: an evaluation study. Proc SigChi 2004. 2004. Vienna: ACM: p. 1337-1340.
30. Plimmer, B. and Mason, R. A Pen-based Paperless Environment for Annotating and Marking Student Assignments, Proc AUIC 2006, CRPIT press.
31. Rubine, D. *Combining gestures and direct manipulation*. proc *CHI '92*. 1992: p. 659-660.
32. UML™ Home Page, OMG (Object Management Group), available from http://www.uml.org/
33. Zhu, N., Grundy, J.C., Hosking, J.G, Constructing domain-specific design tools with a visual language meta-tool', Proc CAiSE 2005 Forum, Portugul, 13-17 June, 2005, p.139-144.