Pen-based Input of UML Activity Diagrams for Business Process Modelling

Alastair F. Donaldson
Department of Computing Science, University of Glasgow, Glasgow, Scotland
ally@dcs.gla.ac.uk

Adrian Williamson
Graham Technology, India of Inchinnan, Renfrewshire, Scotland
Adrian.Williamson@GrahamTechnology.com

We describe the design and implementation of a prototype system for pen-based input of business process models. It is common practice for business processes to be sketched on a traditional whiteboard, using a modelling notation such as UML activity diagrams, during meetings between software developers and business clients, being input to business process management software at a later date from a photograph of the whiteboard. Our system supports modelling with activity diagrams using a pen-based input device such as an electronic whiteboard or graphics tablet. Activity diagram components are specified by single-stroke gestures, which are recognised and immediately replaced by diagram components for further manipulation. The system also provides handwriting recognition for textual annotation of components. Handling input of UML activity diagrams involves dealing with various forms of rich pen-based input including arrows, dashed lines, formatted text, and connections between diagram components. We approach the problem of recognising such input by inferring component features based on the context in which input occurs. Through an informal user evaluation of our prototype, we have identified some problems differentiating between similar activity diagram components, and we discuss potential solutions to these problems. Finally, we describe plans for future work integrating our approach with a proprietary business process management suite.

Pen-based input, business process modelling, UML activity diagrams, gesture recognition, BPMN, BPML.

1. INTRODUCTION

Business process management [1,2] software allows business processes to be specified in a graphical notation such as activity diagrams in the Unified Modelling Language (UML) [3,4], or Business Process Modelling Notation (BPMN) [5]. Once modelled, processes can be executed by a business process management suite (BPMS). Process diagrams are typically input to a BPMS using a drag-and-drop diagramming tool. Before using such a tool, it is common practice for business processes to be sketched on a traditional whiteboard during meetings between software developers and business clients. Process diagrams are then input to a BPMS at a later date, from a photograph of the whiteboard. When sketching initial process designs there are clear advantages to using a whiteboard. It may be convenient to sketch incomplete diagrams, or to temporarily abuse formal notation while brainstorming. Furthermore, whiteboards are suitable for collaborative design. However, there are clear advantages to designing business processes using a software diagramming tool: diagram components can be manipulated easily; diagrams can be saved and reloaded for subsequent editing; and (syntactically correct) diagrams can be executed by a BPMS.

To combine the desirable features of modelling with a whiteboard and with a diagramming tool, we have built a prototype system which supports modelling of business processes using a pen-based input device such as an electronic whiteboard or graphics tablet. Our system uses the UML activity diagram notation, and activity diagram components are input to the prototype via gestures derived from component shapes. Once input, the pen-stroke representing a component is immediately replaced with an icon for that component, which can be manipulated and annotated. The prototype uses a handwriting recognition engine to allow the textual annotation of diagram components. To ease the recognition of rich pen-based input such as arrows, dashed lines, formatted text, and connections between components, our system exploits the context in which input takes place. We have informally evaluated the prototype with software developers. A widely used commercial BPMS includes a design and analysis tool (see Figure 3), for modelling business processes using BPMN or activity diagram notation. Our long term goal is to integrate the functionality of our prototype with this design and analysis tool, to allow pen-based modelling of business processes which can then be executed directly in the suite.

1.1 Paper Structure

We begin by discussing related work in Section 2. In Section 3 we provide a brief summary of the UML activity diagram components which our tool supports. We describe the design and implementation of the prototype in Section 4, and elaborate on the details of gesture recognition, handwriting recognition, and rich pen-based input in Section 5. We make some observations based on informal user evaluation of the prototype in Section 6. Ideas for future work are discussed in Section 7.

2. RELATED WORK

Pen-based input of UML class diagrams has been investigated using *Knight*, a system which allows modelling of class diagrams on an electronic whiteboard [6]. The system operates in two modes: freehand mode, in which doodles and sketches can be input, and UML mode, in which class diagram components can be recognised. Like our system, components are specified as gestures, recognised using Rubine's algorithm [7], and are interpreted as soon as they are drawn. To increase the number of gestures which can be recognised, the system uses compound gestures [8], and *eager recognition* [7] is used to provide feedback while gestures are drawn. The system does not support UML activity diagrams.

The *SUMLOW* (Sketching UML On Whiteboard) system [9] takes a different approach, aiming to retain the look and feel of real whiteboards as much as possible. The system does not automatically replace UML components with icons as soon as they are recognised – it allows components to be manipulated in a sketched form, and progressively formalised when desired. The system supports a variety of UML diagram types, and allows components from different kinds of diagram to be mixed together, even though this may violate the semantics of a particular diagram type. This approach to handling pen input involves the "beautification" of a freehand sketch to obtain a formalised, computer drawn version [10].

A more general system, to provide a front end for on-line recognition of any glyph-based diagram notation, is described in [11]. The kernel of the system has been extended specifically to recognise UML components. Other related systems include the *Software Design Board*, a prototype collaborative design tool which supports the freehand creation of UML diagrams [12]; and an extended version of the *Tivoli* application, allowing the input and rearrangement of material on an electronic whiteboard [13].

3. OVERVIEW OF UML ACTIVITY DIAGRAMS

UML activity diagrams can be used to model dynamic aspects of systems [3] and thus provide a useful notation for describing business processes. Sequential and concurrent steps in a process are represented by *activities*, with flow of control indicated by *transitions*, *branches*, *forks* and *joins*. Data flow and communication can also be modelled using *objects* and *signals*. A full discussion of the role of activity diagrams in relation to other forms of UML diagram can be found in [3], while [4] is a concise notational reference.

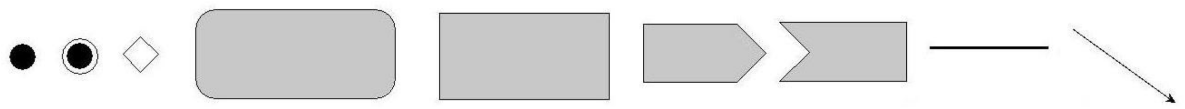

FIGURE 1(A): UML activity diagram components from left to right: start node; end node; branch; activity; object; signal transmission; signal receipt; fork/join; transition.

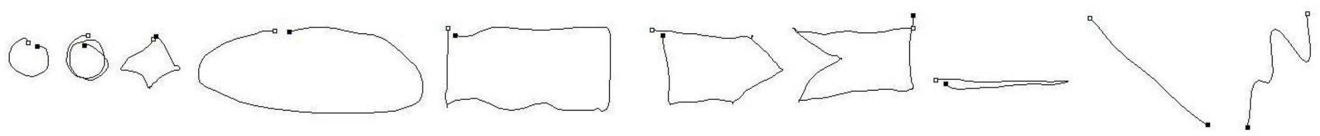

FIGURE 1(B): Gestures corresponding to activity diagram components. Small white and black squares respectively denote the start and end of each gesture. The gesture on the far right is used to switch to handwriting mode (see Section 4.2).

Figure 1(A) summarises the icons used by our prototype to represent UML activity diagram components. The core of any activity diagram consists of a start node, a transition to a sequence of activity nodes, themselves connected by transitions, and a transition from this sequence to an end node. Conditional behaviour can be modelled using branch nodes, and the fork/join component can be used to fork control flow into concurrent threads, and to merge these threads at a later point. Data flow and communication can also be modelled using objects and signals. A dashed line with an arrowhead, omitted from Figure 1(A), is used to indicate data flow. The right hand screenshot of Figure 2 shows an example of an (incomplete) activity diagram for an ordering and shipment process [4].

4. PROTOTYPE IMPLEMENTATION

We implemented our prototype as a Java application, using the JGraph package [14] for the representation and manipulation of diagram components. We now summarise the features which our prototype supports, and discuss some currently unsupported features which we plan to incorporate eventually.

4.1 Main Features Supported

- **Recognition of activity diagram components** Our prototype has been designed to recognise all of the activity diagram components shown in Figure 1(A).
- **Handwriting** In order to allow textual annotation of components, our prototype includes handwriting recognition, relying on a third party handwriting recognition engine.
- **Context-based feature inference** To ease the problems associated with recognition of rich pen-based input, where possible our system has been designed to infer features of diagram components based on their context. This is discussed in detail in Section 5.3.
- **Loading and saving** Activity diagrams can be saved and reloaded for subsequent editing.
- **Training tool** The gesture recognition component of the prototype, discussed in Section 5, is trained by specifying examples for each class of gestures. The system includes a tool to train the recogniser, and this tool can also be used to specify which features of a gesture should be considered during recognition.

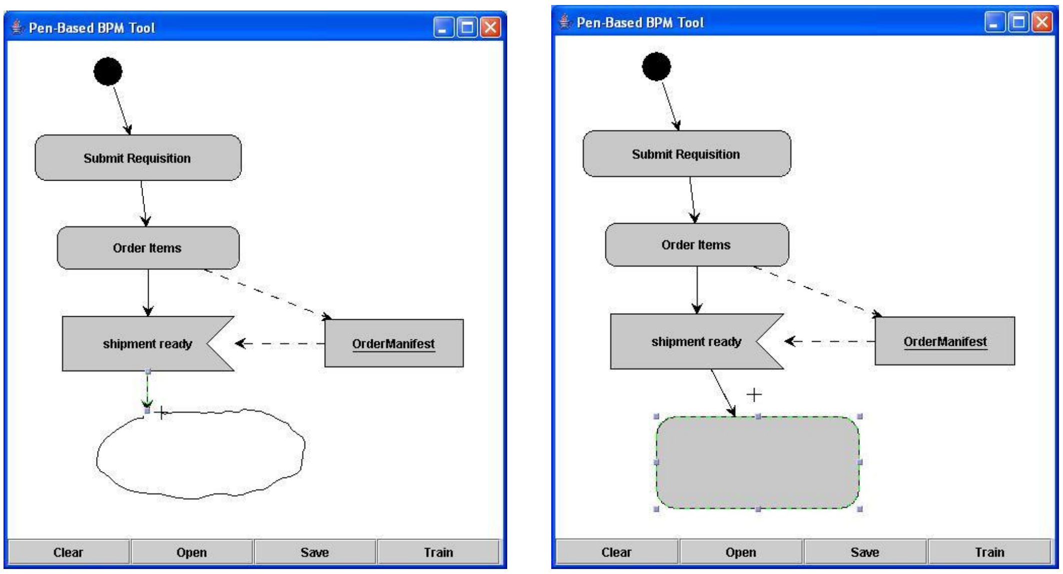

FIGURE 2: These screenshots show our prototype recognising an *activity* node. The screenshot on the left shows the input gesture for the node, which has been replaced by an activity component on the right.

4.2 Unsupported Features

- **Swimlanes** Several *actors* may participate in a business process. *Swimlanes* are used in activity diagrams to indicate which actor performs a given activity in a process. Figure 3 shows an activity diagram with swimlanes for two actors: *Requestor* and *Manager.* To incorporate the functionality of our prototype into a proprietary BPMS we will need to consider the issues involved in pen-based specification of actors and swimlanes.
- **Syntax checking** Our prototype allows users to input diagrams which are *not* correct activity diagrams according to the UML standard. While it is desirable for users to be able to do this (it is common to temporarily abuse formal notation during a brainstorming session), it would be useful to eventually incorporate an option to check whether or not a diagram is syntactically correct.
- **Nested diagrams** Modular modelling of business processes is supported by the UML via *nested* activity diagrams – an activity node may itself contain another activity diagram. Although our tool allows an activity diagram to be drawn inside an existing activity node, we do not yet provide any formal notion of diagram nesting.
- **Formal representation** In order to make the activity diagrams produced by our system usable by existing software for business process management it will be necessary to translate the graphical representation of a diagram to a formal representation such as XML Metadata Interchange (XMI) [15], which provides a standardised

way to exchange models based on the Meta Object Family (MOF) [16]. Such an approach to the formal representation of UML diagrams for pen-based input has already been used in practice [17].

The screenshots of Figure 2 illustrate the operation of the prototype. We now discuss the way in which our prototype recognises activity diagram components and handwriting, and describe how we exploit the context in which input takes place to infer component features for rich pen-based input.

5. RECOGNITION AND ANNOTATION OF ACTIVITY DIAGRAM COMPONENTS

We have implemented a gesture recogniser using a standard feature-based approach [7]. The recogniser can be trained by providing a small number of examples of each gesture. In this section we describe our choice of gestures for input of activity diagram components, discuss the way in which our prototype deals with handwriting, and show that in several cases the task of recognising rich pen-based input can be made easier by exploiting the context in which input takes place.

5.1 Designing Gestures for Activity Diagram Components

The gestures used to specify the various activity diagram components of Figure 1(A) are shown in Figure 1(B). There is an additional gesture on the right of Figure 1(B) which is used to switch to handwriting mode (see Section 5.2). Where possible, we have tried to use the shape of a component as the gesture which it is represented by. This ensures that our system is intuitive to use, and minimises the number of specialised gestures which users must learn. There are two exceptions to this rule. The *end node* component, represented by a double circle, cannot be drawn in one stroke. The gesture we have chosen for this component involves two traversals of a circle in a single stroke, simulating the act of drawing two circles. The *fork/join* component, represented by a thick line, is specified by drawing a line which doubles back on itself. This simulates the act of drawing a bold line, and avoids confusion with the gesture for a transition between components. Since both of these specialised gestures have been derived from the components which they represent, they should be easy to remember. The arrow head associated with a transition is omitted from its gesture – this is explained in Section 5.3.

5.2 Handwriting Recognition for Textual Annotation of Components

To deal with handwritten text, our prototype switches to *handwriting* mode. Once this mode is entered, all pen strokes are stored, and displayed on screen, until the mode is exited. At this point, the stored pen strokes are passed to a handwriting recognition engine, and the text returned by this engine is displayed with appropriate formatting (see Section 5.3), replacing the initial pen strokes.

The problem of mode switching is a key issue in the design of pen interfaces. The challenge is to find a technique for mode switching which is both fast and reliable. In order to be consistent with the input method for diagram components, we decided to use a “scribble” gesture, shown on the right of Figure 1(B), to enter handwriting mode, rather than a button or menu option. With this approach, the user is required to remember an additional gesture, and the gesture recognition system has an extra gesture which must distinguish from other gestures. To overcome the first of these problems we designed the mode switching gesture as a scribble to intuitively represent handwriting. To minimise the second problem we made the gesture significantly different from all other gestures. Our small evaluation, described in Section 6, indicates that this approach to mode switching works well, although a larger evaluation will be required to draw any definite conclusions.

Once in handwriting mode, and after writing has commenced, a countdown timer is reset at the end of each pen-stroke. If the countdown times out before the next pen-stroke then handwriting mode is exited. The advantage of this approach is that no explicit action is required to exit handwriting mode. Possible disadvantages are that the countdown may time out if the user pauses before they have finished writing; the user may resume the input of diagram components *before* the countdown has timed out; or the user may simply be frustrated that it takes too long to exit handwriting mode. We experimented with various countdown lengths, and found a countdown of a second to be practical. Using visual feedback to show the state of the countdown timer could potentially reduce the problems associated with this approach.

A variety of explicit mode switching techniques are described and analysed in [18]. These include: pressing the barrel button of the pen; pressing and holding the pen still; pressure-based mode switching; using the eraser end of the pen; and using of the non-preferred hand, to switch mode. The analysis shows that using the non-preferred hand to switch modes is the fastest of these techniques, while pressing and holding the pen still is slow and error-prone. It would be interesting to compare our approach of using a gesture and timer to switch between modes with these existing techniques.

We also experimented with automatic entry of handwriting mode: if a pen stroke could not be classified as a component gesture then it would be assumed to be the start of some handwriting. This approach is an example of *mode inferencing*, where input is analysed in an attempt to automatically switch modes according to the user's intention. Although in some cases our mode inferencing approach worked well, we found the input of text beginning with certain letters problematic if the letter could be interpreted as a component gesture. For example, the letter 'o' is hard to distinguish from the *start node* gesture. Another problem arose when an intended component gesture was interpreted as text, causing unintentional entry of handwriting mode. This illustrates the inherent ambiguity of pen-based input due to overloading of the limited input modality of a stylus, a problem which is discussed in detail in [19].

5.3 Exploiting Context for Rich Pen-Based Input

Any pen-based input technique for activity diagrams must be able to handle rich pen-based input, including arrows, which indicate the direction of control/data flow; dashed lines, which distinguish data flow from control flow; and connections between components, used to indicate the diagram nodes involved in a transition. It should also be possible to support the formatting of textual input according to UML naming conventions. Such rich pen-based input is difficult to recognise directly if components are to be input as single stroke gestures. We now describe how we avoid this recognition problem by exploiting the context in which input takes place.

Connections between components When an edge is added to a diagram via a line gesture, it is straightforward to determine from the context of input whether or not the edge is meant to connect two diagram nodes. If the start of the edge occurs inside or close to a node then that node is taken to be the source of the edge. The target of the edge is determined similarly from the end of the edge.

Arrows Activity diagram components are connected by directed edges to represent flow of control and data. The direction of an edge indicates the direction of the flow. Typically, a directed edge would be drawn in two strokes – one for the line of the edge, another for the arrow head – which does not fit our approach of interpreting components using single-stroke gestures. We solve this problem by using an *undirected* line gesture to specify an edge, and infer the direction of the edge from the direction of the gesture.

Control flow vs. data flow Transitions between activities (and signals) in an activity diagram are represented by unbroken lines, while dashed lines are used to indicate the flow of data between an object and an activity or signal. Specifying a dashed line using a single stroke gesture is problematic. Rather than solving the problem of handling input of dashed lines, the style of a line is again determined from the context in which the line gesture occurs. If the start or end of a line connects to an object node then the dashed line style is set, otherwise the default line style is used. Additionally, if a line gesture occurs in a context which is not allowed (e.g. a line which starts in an end node and finishes in an activity node), the line colour is set to red, indicating that there is an error. Line styles are refreshed each time a component is added to, or deleted from the diagram.

Formatted text When writing UML activity diagrams, it is usual to stick to standard conventions on the format of text associated with components. For example, the name of a guard associated with a transition should be enclosed in square brackets (e.g. [accept]); the name of an object should contain no spaces, with capital letters for new words, and should be underlined (e.g. <u>RequisitionReciept</u>); and similar conventions apply to activities and signals. It may be hard to stick to these conventions when using a handwriting recogniser. For example, square brackets enclosing a guard name, or a line beneath an object name, may be interpreted as part of that name. Since handwriting recognisers use spell checkers to aid recognition of words, the concatenation of two words representing the name of an object may be interpreted as a misspelling of a different, single word, resulting in the correct spelling of this single word being returned by the recogniser. We avoid this problem by allowing the user to ignore such conventions, inputting text as separate words and omitting features such as square brackets and underlining. Once the text has been recognised it is processed according to the context in which it is input, and an appropriate convention is applied. For example, if text is input over an object then the first letter of each word will be capitalised, all spaces will be removed, and the text will be displayed with underline style. We could provide options for users to specify their own formatting conventions, and use an undo-stack to allow automatic formatting operations to be cancelled if desired.

6. EXPERIENCES TESTING THE PROTOTYPE WITH USERS

We have yet to conduct a formal user evaluation of our prototype. However, initial testing of this prototype with a small group of software developers has led to some interesting observations which we now note.

The five developers who we observed using the prototype had little or no experience in operating a pen-based input device. Using a graphics tablet to operate the prototype, we allowed each user to train the gesture recogniser by providing a series of example gestures, then asked them to transcribe a short activity diagram, provided on paper. Users found that the recognition algorithm, trained with their own example gestures, performed rather poorly, frequently displaying a UML component differing from the intended input. We then asked each user to input another diagram, initialising the recogniser with a training set of gestures provided by the prototype implementer.

The difference in recognition with this training set was dramatic, exceeding the improvement that would be expected from the users' increased experience on their second attempt using the tool. In general, through our preliminary experiments, we have found that our gesture recogniser performs best when trained with a small set of accurate gestures, supplied by an experienced user of pen-based devices. Users had no problems switching to handwriting mode using the approach described in Section 5.2.

Despite the improved rates of recognition when using a training set provided by an experienced user of the system, we found that that users still had some difficulty with the input of *objects* (sharp rectangles), which would often be recognised incorrectly as *activities* (rounded rectangles), or *signals* (concave and convex pentagons). Such errors clearly occur due to the similarity between the shapes used to represent these components. Currently our recogniser assesses input gestures using a set of thirteen features suggested in [7]. We intend to examine the values for these features for examples of *object*, *activity* and *signal* gestures, to see if the gestures are significantly distinguished by particular features. If so, we may be able to improve recognition by restricting the recogniser to work only with these distinguishing features, as long as this restriction does not have a significant detrimental effect on the recognition of other gestures. The problem of differentiating between similar gestures is considered in [20], in which the *quill* tool [21,22] is used both to warn designers of gestures which may be too similar to be reliably differentiated, and provide advice on how to modify gestures to be less similar. We may be able to make use of this work to improve recognition in our system, although it is desirable for gestures to be similar to the shapes they represent so that the system is intuitive to use. We could also consider *compound gestures* [8], which have been used to increase the number of distinct gestures which can be recognised reliably [6]. Currently our prototype relies solely on gesture recognition to identify input shapes. Combining gesture recognition with algorithms for shape recognition (e.g. [23]) may help to improve differentiation between similar component shapes.

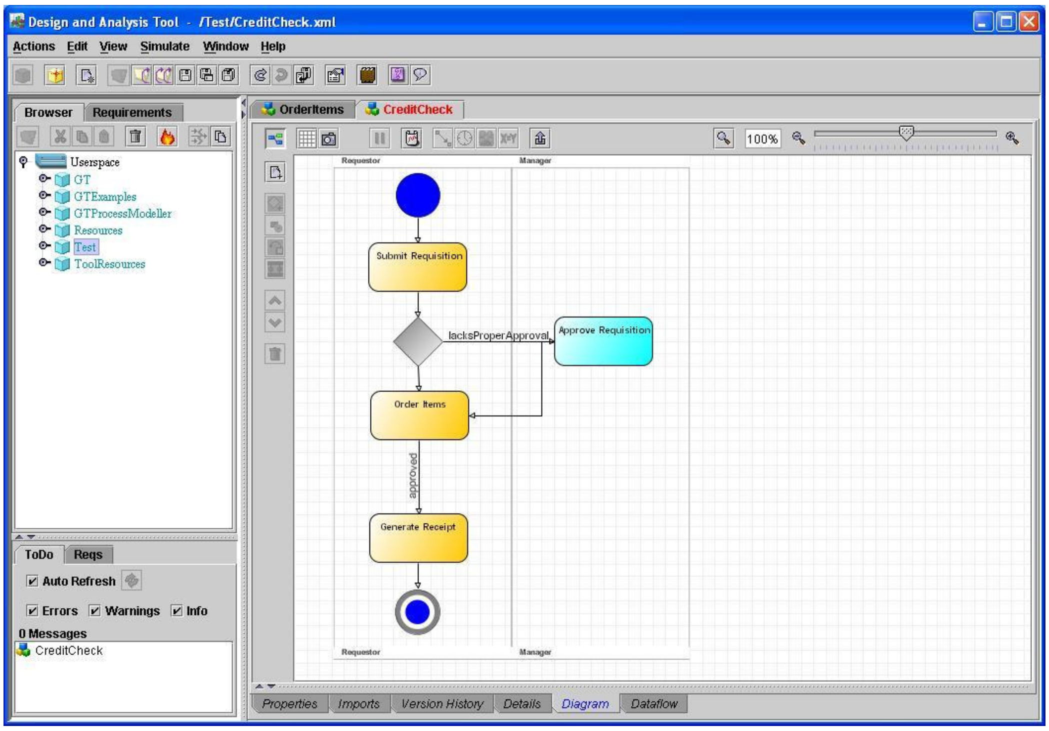

FIGURE 3: Modelling an ordering process using a commercial BPMS.

7. CONCLUSIONS AND FUTURE WORK

We have presented a prototype system for pen-based modelling of business processes as UML activity diagrams. Our system supports the main activity diagram components, allows textual annotation of components, and handles rich pen-based input by exploiting the context in which input takes place. We plan to extend the prototype to include the unsupported features described in Section 4.2., and solve the recognition problems identified by our informal evaluation, discussed Section 6. The functionality of our prototype, together with ideas for improvement identified by a formal evaluation, will be the basis for incorporating pen-based modelling of business process diagrams into the design and analysis tool of a proprietary BPMS, of which a screen shot is provided in Figure 3. This tool currently supports BPMN as well as UML activity diagrams, so we will need to extend our current gesture set accordingly for BPMN. The prototype currently operates by immediately interpreting input gestures, and replacing pen-strokes with recognised objects, in a similar manner to *Knight* [6]. It would be interesting to experiment with progressive formalisation of diagrams, in the spirit of *SUMLOW* [9], using a *recognise* gesture to tell the system to update all pen-strokes with recognised components.

ACKNOWLEDGEMENTS

This work was supported by Graham Technology plc., and the proprietary BPMS referred to in the paper is GT-X [24]. For recognition of handwritten text, the prototype system uses the Microsoft XP handwriting recognition engine [25].

REFERENCES.

[1] Business Process Management Initiative (2005) *Home Page* [WWW document] URL: http://www.bpmi.org/ (visited 11th August 2005).

[2] Object Management Group OMG *Home Page* (2005) [WWW document] URL: http://omg.org/ (visited 11th August 2005).

[3] Booch, G., Rumbaugh, J., and Jacobson, I. (1999) *The Unified Modeling Language User Guide*. Addison-Wesley.

[4] Pilone, D. (2003) *UML Pocket Reference*. O'Reilly.

[5] Object Management Group, Business Process Management Initiative (2004) *Business Process Modelling Notation (BPMN) Information* [WWW document] URL: http://www.bpmn.org/ (visited 11th August 2005).

[6] Damm, C.H., Hansen, K.H., and Thomsen, M. (2000) Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard. *Proceedings of CHI 2000*, The Hague, The Netherlands, 1–6 April, pp. 518–525. ACM Press.

[7] Rubine, D. (1991) Specifying Gestures by Example. *Computer Graphics,* **25(4)**, 329–337.

[8] Landay, J.A., and Myers, B.A. (1995) Interactive Sketching for the Early Stages of User Interface Design. *Proceedings of CHI 95*, Vancouver, Canada, April 14–18, pp. 45–50. ACM Press.

[9] Chen, Q., Grundy, J.C., and Hosking, J.G. (2003) An E-whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams. *Proceedings of HCC 2003*, Auckland, New Zealand, 28–31 October, pp. 219–226. IEEE Computer Society Press.

[10] Plimmer, B., and Grundy, J. (2005) Beautifying Sketching-based Design Tool Content: Issues and Experiences. *Proceedings of AUIC 2005*, Newcastle, Australia, 31 January – 3 February, Conferences in Research and Practice in Information Technology 40, pp. 31–38. Australian Computer Society.

[11] Lank, E., Thorley, J., Chen, S., and Blostein, D. (2001) On-Line Recognition of UML Diagrams. *Proceedings of ICDAR 2001*, Seattle, Washington, September 10–13, pp. 356–360. IEEE Computer Society Press.

[12] Wu, J., and Graham, T.C.N. (2005) The Software Design Board: A Tool Supporting Workstyle Transitions in Collaborative Software Design. *Proceedings of EHCI-DSVIS 2004*, Hamburg, Germany, 11–13 July, Lecture Notes in Computer Science 3425, pp. 363–382. Springer.

[13] Moran, T.P., Chiu, P., and Melle, W.v. (1997) Pen-Based Interaction Techniques for Organizing Material on an Electronic Whiteboard. *Proceedings of UIST 1997*, Banff, Alberta, 14–17 October, pp. 105–114. ACM Press.

[14] JGraph (2005) *Open Source Java Graph Visualization and Layout* [WWW document] URL: http://www.jgraph.com/ (visited 11th August 2005).

[15] Object Management Group (2002) *OMG XML Metadata Interchange (XMI) Specification* [WWW document] URL: http://www.omg.org/docs/formal/02-01-01.pdf (visited 11th August 2005).

[16] Object Management Group (2000) *Meta Object Family (MOF) Specification* [WWW document] URL: http://www.omg.org/docs/formal/00-04-03.pdf (visited 11th August 2005).

[17] Laird, C. (2001) *XMI and UML combine to drive product development* [WWW document] URL: http://www-128.ibm.com/developerworks/xml/library/x-xmi/ (visited 23rd August 2005).

[18] Li, Y., Hinckley, K., Guan, Z., and Landay, J.A. (2005) Experimental Analysis of Mode Switching Techniques in Pen-based User Interfaces. *Proceedings of CHI 2005*, Portland, Oregon, 2–7 April, pp. 461–470. ACM Press.

[19] Deming, K., and Lank, E. (2004) Managing Ambiguous Intention in Mode Inferencing. *Proceedings of the AAAI Fall Symposium Series: Making Pen-based Interaction Intelligent and Natural*, Washington DC, 21–24 October, pp. 49–54. AAAI Press.

[20] Long, A. C., Landay, J. A., and Rowe, L. A. (2001) "Those Look Similar!" Issues in Automating Gesture Design Advice. *Proceedings of PUI 2001*, Orlando, Florida, 15-16 November. ACM Digital Library.

[21] Long, A. C. (2001) *Quill: a Gesture Design Tool for Pen-based User Interfaces*. PhD dissertation, University of California at Berkeley, Berkeley, California.

[22] Long, A. C., Landay, J. A., and Rowe, L. A. (1999) Implications for a gesture design tool. *Proceedings of CHI 1999*, Pittsburgh, Pennsylvania, May 15–20, pp. 40–47. ACM Press.

[23] Chang, C.C., Hwang, S.M., and Buehrer, D.J. (1991) A Shape Recognition Scheme Based on the Relative Distances of Feature Points from the Centroid, *Pattern Recognition*, **24(11)**, 1053–1063.
[24] Graham Technology (2005) *GT-X 7* [WWW document] URL: http://www.grahamtechnology.com/Home/Products/GTX7.jsp (visited 11th August 2005).
[25] Microsoft (2005) *Handwriting Recognition Overview* [WWW document] URL: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/input_pen_overview.mspx (visited 11th August 2005).