Restructuring Data Constructs in Overlapping Digital Ink Domains for Agent-Oriented Approaches

K. A. Mohamed, Th. Ottmann
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, 79110 Freiburg, Germany
{khaireel, ottmann}@informatik.uni-freiburg.de

Abstract

We examine the common hierarchical data structure of digital ink representations in two distinctive ink domain-classes. Digital boards (eBoard) can receive inputs for both archival and reactionary modes from freehand writings (or drawings) and gesture-commands, respectively, from the digital pens. The latter is modelled as a result of the former in theoretical pattern recognition, and perhaps, more for convenience. What seems to be hypothetically sound, this structural data construct appears limited when attempting to succinctly decipher ink inputs as either traces or command gestures when both domains are merged on a single platform environment. This report forestalls a possible method of improvement to integrate both the 'sketching' and 'gesturing' domains simultaneously on the eBoards. Our proposed change to the data structure tailors for agent-oriented approaches and allows for parallel evaluation with very little overheads. It also provides a basis to address the problems of ambiguity when attempting to distinguish the exact domain from an input ink data.

Keywords: Agent oriented software engineering, digital ink, pen gestures, InkML, learning environments
CR Classification: G.1.10, H.3.3, I.7.5

1. Introduction

The Perceptual User Interfaces (PUI) paradigm, as discussed by Turk and Robertson [21] and Landay et al. [11], attempts to do away with conventional GUIs, and instead, pushing towards other *perceptual* starting points in the design space, bringing about the concept of "invisible computers". Many modern and networked learning environments utilise the electronic pen's digital ink for a convenient way to visually communicate ideas within the vicinity of digital screens. This method of communication is seen as a natural way of exchanging viewpoints; its a digital upgrade, if you like, from the messy blackboards and chalks, to the cleaner whiteboards and markers, to the *cooler*, state-of-the-art eBoards and styluses. This trend of moving towards natural environments in the digital world is what encourages the development of PUI-based programs to match the techniques of how people use their computers.

We want to reproduce the simple, customary blackboard, and still be able to include all other functionalities that an eBoard can offer. But by getting rid of the static menus and buttons (in accordance to the PUI standards), the resultant 'clean' slate becomes the only perceptual input available to users to relate to the background systems. Here, we see two distinct domains merged into one – the domain to receive handwritings (or drawings) as symbolic representation of information (termed technically as *traces*), and the domain to react to user commands issued through pull-down menus and command buttons.

Based solely upon the input ink traces, we must now be able to decipher users' intentions in order to correctly classify which of the two domains it is likely to be in (either as primitive symbolic traces, or some sort of system commands). There are currently many works by authors that describe vividly the interpretations of these traces – 'exclusively' in either domains, as well as in the 'combination' of the two. We agree that incorporating pen-based command *gestures*, as a further evaluation of the input traces, as an alternative to issuing system commands, for addressing this scenario, is indeed one of the most practical ways to solve the new paradigm problem.

Unfortunately, the published results appear somewhat limited – such that toolkits developed can only accommodate a certain number of gestures before serious ambiguity problems arise [18], or that users can only gesture at designated portions of the screen [4], or that users must learn special escape sequences to manually switch between the two domains [8]. The main circumspection of these limitations, it seems, lies in the management and utilisation of the underlying data structures. The present ones may not suit the overlapping of the two domains efficiently enough so as to preserve the *naturality* of the interaction within the eBoard

environments.

In the course of this report, we will showcase the fundamentals of the current data structures, in general, and explain why they appear bounded in PUI scenarios.

We first highlight some of the digital ink concepts and their applications in the next section. Following which, we bring forth the general data structures that are currently used to represent both ink traces and ink gestures. Section 4 then argues whether or not a trace should be considered a gesture, which leads to the conception of our data structure, detailed in Section 5. The next sections give an example of how we use the new data construct in an agent-oriented environment and contrasts our preliminary experimental results of the degree for efficiency and autonomy when compared to the old structure.

2. The Digital Ink

Our review of some of the more prominent digital ink literature shows numerous progressive works where most authors dwell specifically in their own domain-related scenarios. There is an obvious segregation of ideas and implementations that center specifically around both domains, as we described in Section 1, as well as an abundance of materials addressing the overlapping of the two. Figure 1 demonstrates this idea graphically.

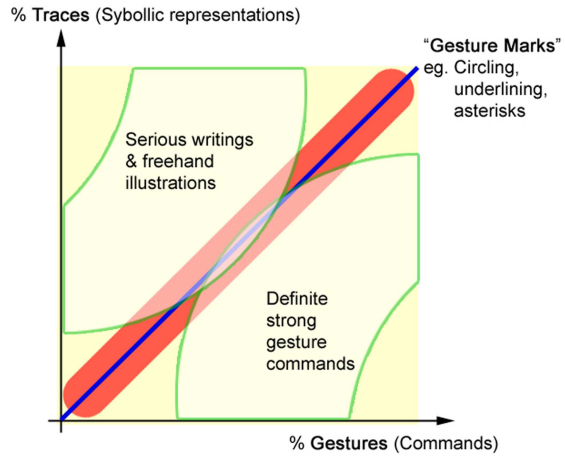

Figure 1. Mapping digital ink in two domains – *traces* or *gestures*.

2.1. In Separate Domains

In the *trace*-only domain, Lopresti et al.'s collective research in dealing with a concentrated area of deciphering digital inks as hand-drawn sketches and hand-writings, and then being able to perform pictorial queries on them, is the result of their effective categorisation of ink as first-class data type in multimedia databases [2, 15, 14]. Others like Bargeron and Moscovich [3] and Goetze et al. [8] analyse the users' rough annotations and open-ended ink markings on formal documents and then provided methods for resetting these traces in a more orderly, cross-referenced manner.

While on the other perspective, we see pilot works on pen *gestures*, which began even before the introduction of styluses for the digital screens. They are purported on ideas of generating system commands from an input sequence of predetermined mouse-moves [19]. Moyle and Cockburn built simple gestures for the conventional mouse to browse web pages quickly, as users would with the digital pen [16]. As gesturing with the pen gained increasing popularity over the years, Long et al. described an exhaustive computational model for predicting the similarity of perceived gestures in order to create better and more comfortable user-based gesture designs [1].

2.2. In a Common Domain

For reasons of practicality and application-suitability, but not necessarily the simplicity of implementation, well-developed toolkits inter-combine the pen input modality for two modes; sketching and gesturing. As we mentioned earlier, automatic classification of ink inputs directed for either modes do not usually include too many gestures, and these tools normally place heavier cognition loads on the sketching mode.

Li et al.'s 'SketchPoint' is an example that recognises only five gestures [12]. The toolkit allows for sketching presentation outlines with handwritings and drawings, and then organising the flow with gestures such as 'listing' and 'inserting' slides. Other of such similar toolkits includes 'Flatland', 'SATIN', 'DENIM', 'Teddy', the 'Cocktail Napkin' and many others [17, 9, 13, 10, 5].

However, one important point to note here is that all of the above toolkits still provided pull-down menus and command buttons, *conspicuously*, for the interaction between users and the system, should the gesture recognisers fail to deliver the desired requests. We see it as a backup plan while still working towards the PUI paradigm.

3. The Data Structure in General

We note from all the descriptions of the above works, the related underlying data structures follow a general structural construct built upwards from the primitive *Trace* class, which in turn is simply an extension of a class that can accommodate time-stamped xy-coordinates. Essentially, a *Trace* is made up of joining a string of these (x, y, t) co-

ordinates from a 2D plane with a line. W3C's InkML standard best describes this fundamental representation of the *Trace* [20].

As most pen *Gestures* are thought to be an instance of a *Trace*, we can then depict the *Trace-Gesture* structural construct as shown in Figure 2. This is directly proportional to the theoretical pattern recognition overview mentioned in Gonzales and Woods [7], where a Gesture-pattern is viewed as a direct subset of a Trace-pattern.

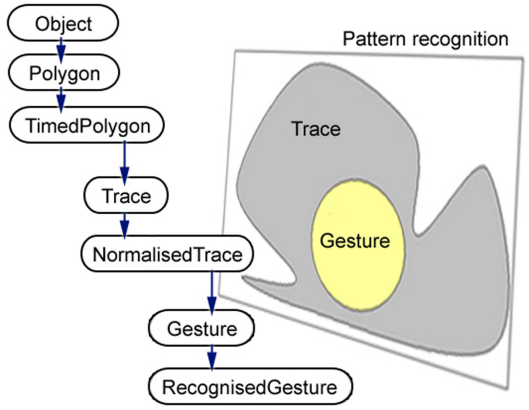

Figure 2. Object-oriented hierarchical structural construct of a *Trace* and a *Gesture*.

On observing the structural construct in Figure 2, we note the following:

- that a *RecognisedGesture* comes a long way from its primitive *Trace* origin;
- that a *Gesture* is completely dependent on a *Trace* instantiation;
- that processing all information is serial; and
- that the overheads in the final sub-class is very large.

In the combined domain, this structural construct may prove to be a burden to any recogniser engines when trying to efficiently classify a pen input as either a *Trace* or a *Gesture* class, without having to perform special escape sequences. There is a fine line of ambiguity between the two classes that needs to be identified and addressed, without compromising to users with what may appear to be *unnatural* delays.

This phenomenon is a reflection on Figure 1, where the overlapping of two domain classes occur. The region labelled 'Gesture Marks' are what many find to be the ambiguous (and indeed problematic) area [5]. For example, a program may wrongly interpret the handwritten alphabet 'O' as a *RecognisedGesture* for invoking a 'select' command. Fortunately, this problem can be solved if the program can anticipate the intentions of its users [6, 25]; however, this method necessitates the constant tracking of the perceptual environment, and would require a more stringent and somewhat 'parallel' structural construct in order to run efficiently.

4. Are Gestures Traces?

There are several interpretations to this question. We base our arguments on the previous sections to justify our answers. We say that a *Gesture* should not be a *Trace* because;

- it is distinctively a "command" to be processed immediately;
- it should not leave marks on the sketching environment;
- there is no need to store a gesture;
- it is 'interactive' while a *Trace* is 'manipulative';
- it may (or may not) affect a *Trace*; and
- it should not be dependent on a *Trace*.

Wexelblat pointed out the difference between gesture recognition and gesture analysis [23]. It is entirely up to the interpreter program to extract the meaning from the inputs and application contexts. If we do not consider gesture recognition as part of pattern classification, we then need to separate the conventional *Trace-Gesture* structural construct (Figure 2). Recognition, we know, is done from the rigourous training of weighted-coefficients of *Gesture*-like features [19]. Analysis refer to the keen observation of those coefficients, and the deciding factor lies in the strength of the final computed variables and, in our case, the probability of the user's intention.

By breaking the *Gesture* from its *Trace* ancestry, we will be able to impose a greater autonomy on both domain classes. It will allow for a more efficient two-way 'anticipatory' interaction between users and the system, while operating on a common input environment.

5. The Constitution of Gesture Commands

Figure 3 demonstrates the process for erasing traces by using gesture commands. Where at time t_0, a user completes gesturing to the eBoard with the intention to erase a previously sketched trace. Most toolkits' interpreters will analyse the new input, and then recognise correctly that it is an 'erase' *Gesture*, and at time t_1, execute the erase command.

If we take a *Gesture* to be a *Trace* essentially, we then see the effect of the command gesture's 'imprint' being left on the sketching environment during this transitional period.

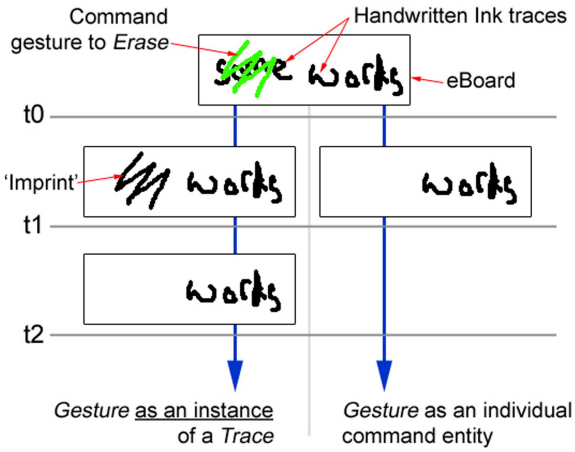

Figure 3. The process of erasing ink traces with *Gestures*.

For this, the background program will have to take an additional step at time t_2 to remove the 'imprint', so as to portray a smooth operation. However, if we perceive a *Gesture* to be an entity all by itself, we will then not see this extra step happening.

5.1. Autonomous Entities

Based on the hierarchical structure depicted in Figure 2, and from the simple example above, we can make the following assumptions:

- a *Gesture* is always a *RecognisedGesture*, otherwise it remains a *Trace*;
- a *Gesture* is stored only for comparing purposes;
- once a *Trace* is upgraded to a *Gesture* status, it immediately cease to exist; and
- both *Gesture* and *Trace* are (autonomously) independent of each other.

5.2. Relating Gestures to Traces

Rubine's linear recogniser algorithm [19], for classifying *Gesture* classes into the various categories within a collection, provides a useful insight of managing an indirect relationship between *Gestures* and *Traces*. Figure 4 sums this up pictorially.

We can extract *features* from an input trace to obtain measurable values of the numerous physical properties attached to a *Trace*. These include information such as angles between sampled points, lengths and speed of the sketched *Trace*. We denote this as $\mathbf{f} = \{f_0, f_1, ..., f_n\}$, where n refers to the number of features extracted.

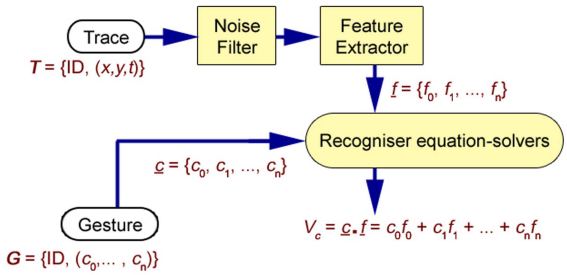

Figure 4. Mathematical relationship between a *Gesture* and a *Trace*.

The linear classifier algorithm is a straight forward dot product of the trained coefficients $\mathbf{c} = \{c_0, c_1, ..., c_n\}$ and the extracted features $\mathbf{f}$. This gives a value V_c that describes the strength of a *Gesture* class c based on the current evaluated features. Here, the strongest V_c is the result of identification process [19].

We note that this *Gesture* class is now simply made up of the trained coefficients $\mathbf{c}$ with its own unique identification tag, and nothing else. We also note that this way of managing the overall information (Figure 4), and applying our assumptions in Section 5.1, that the *Gesture* class is no longer an instance of the *Trace* class.

6. The Separated Data Structure

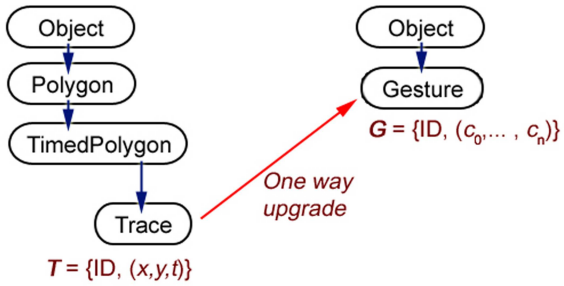

Figure 5. The separated and clean-up structural constructs.

By way of mathematical reasoning highlighted in Figure 4, we are also able to maintain the integrity of each of the object classes, without losing any important information, even if both these classes are perpetually segregated. The corollary of which is a significant reduction of overhead inscriptions for the *Gesture* class.

The separation also makes some classes in the original structural construct unnecessary. These are removed and the

newly separated data structures are shown in Figure 5. Because of this, we can now process any user inputs in parallel, thus making it possible to test two domains simultaneous and, to a certain extent, more efficiently. This brings about a rather typical agent-oriented approach, where we can task two autonomous agents to process the input trace in a commonly overlapped domain.

We mentioned before that if a program is able to intelligently anticipate the intention of their users, through the constant tracking of the perceptual input environment, then the problem of ambiguosity within the overlapping domains can be overcome. An *agency* not only supports this backbone process practically, but also accord them in a general manner that is applicable to an array of other proverbial domains.

7. The Agency Architecture

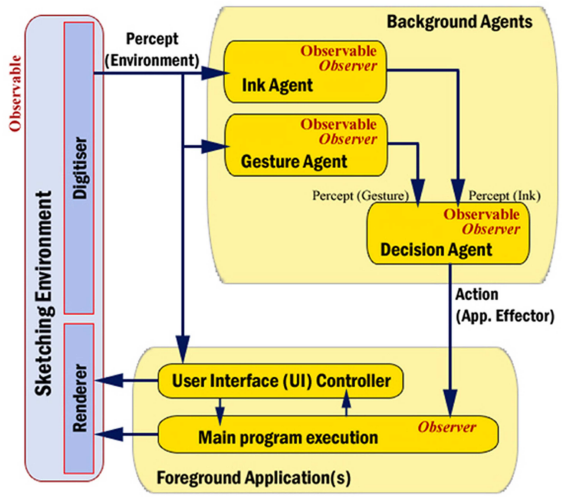

Figure 6. Overall integrated system architecture.

We implemented the architecture as show in Figure 6, to allow the agents to react whenever there is a change occurring within the sketching environment – either affected by users with ink-traces, or by foreground applications. Those input traces are then processed by the Ink and Gesture agents to formalise the data representations and to extract gestural features, respectively, before making the refined information available to any other observers (both agents and application programs) down the chain of the interaction network.

The new data structures depicted in Figure 5 are utilised within the agency, with the Ink agent handling the *Trace* classes and the Gesture agent handling the *Gesture* classes. This architecture provides the necessary 'natural' feedback loop between the agency, the user and the foreground application, to effect the process of *anticipation*. These agents perceive their user-input environment, and then act upon that environment by directing automatic assumptions to the application program or as suggestions for the user to make further decisions. Their means of acclimatic reasoning and goal-oriented aims make them useful assistants when processing in the background as "interface agents" [24, 22].

8. Preliminary Evaluations

8.1. Experimental Setup

As naturally as possible, we simulated the sketching environment to act *exactly* like the digital version of whiteboards used in lecture halls and tutorial classes. Without adding too much details about the pen attributes or the canvas' rendering properties, we concentrated on the issues of our agents assisting tutors delivering "stress-free" lessons. Minimising the trips to pick up board erasers is the main objective of this evaluation, and that substituting preferred gestures to the button clicking of the digital pen is the other aspect we want to observe.

8.2. Observation

The agents are always attentive to the inputs received from the whiteboard canvas. The agency recognises whenever a trace is expected as a gesture, or when it is to remain as ink. The foreground program reacts to the agency's verdicts and churns out a popup menu with the gesture's generic name whenever appropriate, see Figure 7, thus confirming the agents' proactive anticipation.

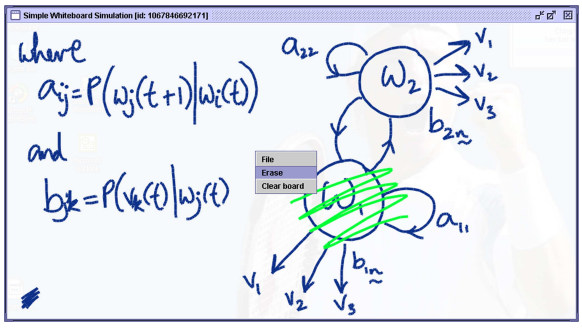

Figure 7. Simple interactive whiteboard assistant simulation.

8.3. Preliminary Results

Our society of agents are able to proactively anticipate about 97% of all true positives from the users' intentional traces, through the Pereira et al.'s *expectation list* technique of handling ambiguity and errors in traces [18]. Figure 8 tabulates the coherence of the Whiteboard assistant, showing the relationship between a user's intention and the agency's collaborative anticipation after 10 whiteboard sessions. In total, users scribed 849 traces on the whiteboard, of which 798 were intended as ink traces and 51 were supposedly gestures. The agency anticipated 788 of the intended ink traces and 49 of the intended gestures correctly. This averages to 97.42% 'true positive' assistance.

Whiteboard assistant - Coherence test				
User's intention	Ink traces: 798		Gestures: 51	
Agency anticipation	788	10	49	2
Percentage %	98.75%	1.25%	96.08%	3.92%
	Correct	Incorrect	Correct	Incorrect

Figure 8. Tabulated coherence of the Whiteboard assistant.

Furthermore, the responses from this whiteboard program are immediate to users. This means that both the foreground application and the background interface agents handle ink information in a good and coordinated manner. It does not give rise to any confusion for the user when interacting with the program. On average, it takes less than 20 msec for the agents to process a trace and then synchronise with the Observer program.

9. Conclusion

We reviewed the conventional structural data constructs for representing digital ink and found that it appears to be limited when two distinct ink domains are merged. In order to stay abreast with the PUI paradigm, it is important that programs and toolkits developed be able to address this merger as efficiently as possible, while maintaining the *naturality* of all human-computer interactions. Our restructuring of the data structures helps to expedite and parallelise both *Traces* and *Gestures* interpretations. On top of which, we the problem of ambiguity is conveniently addressed by supplanting an interactive agency thought the agent-oriented approach.

Acknowledgement

This research is funded by the Deutschen Forschungsgemeinschaft (DFG) as part of the research initiative for the "Algorithmen und Datenstrukturen für ausgewählte diskrete Probleme (DFG-Projekt Ot64/8-3)".

References

[1] J. A. Chris Long, J. A. Landay, L. A. Rowe, and J. Michiels. Visual similarity of pen gestures. In *Proceedings of the CHI 2000 Conference on Human Factors in Computing Systems*, pages 360–367. ACM Press, 2000.

[2] W. G. Aref, D. Barbará, D. Lopresti, and A. Tomkins. Ink as a first-class datatype in multimedia databases. *Multimedia Databases*, pages 113–141, 1995.

[3] D. Bargeron and T. Moscovich. Reflowing digital ink annotations. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 385–393. ACM Press, 2003.

[4] S. Chatty and P. Lecoanet. Pen computing for air traffic control. In *Conference Proceedings on Human Factors in Computing Systems*, pages 87–94. ACM Press, 1996.

[5] E. Y.-L. Do. What's in a diagram (that a computer should understand)? *Computer Aided Architectural Design Futures '95*, pages 469–482, 1995.

[6] E. Durfee, V. Lesser, and D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, KDE-1(1):63–83, March 1989.

[7] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Pub Co, 2nd edition, 2002.

[8] M. Götze, S. Schlechtweg, and T. Strothotte. The intelligent pen: toward a uniform treatment of electronic documents. In *Proceedings of the 2nd International Symposium on Smart Graphics*, pages 129–135. ACM Press, 2002.

[9] J. I. Hong and J. A. Landay. Satin: a toolkit for informal ink-based applications. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 63–72. ACM Press, 2000.

[10] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 409 – 416. ACM Press, July 1999.

[11] J. A. Landay, J. I. Hong, S. R. Klemmer, J. Lin, and M. W. Newman. Informal PUIs: No recognition required. In *Proceedings of AAAI 2002 Spring Symposium (Sketch Understanding Workshop)*, pages 86–90. AAAI Press, 2002.

[12] Y. Li, J. A. Landay, Z. Guan, X. Ren, and G. Dai. Sketching informal presentations. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, pages 234–241. ACM Press, 2003.

[13] J. Lin, M. W. Newman, J. I. Hong, and J. A. Landay. Denim: Finding a tighter fit between tools and practice for web site design. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 510–517. ACM Press, 2000.

[14] D. Lopresti and A. Tomkins. Temporal domain matching of hand-drawn pictorial queries. *Handwriting and Drawing Research: Basic and Applied Issues – conference version in The 7th Biennial Conference of The International Graphonomics Society*, pages 387–401, 1995.

[15] D. Lopresti, A. Tomkins, and J. Zhou. Algorithms for matching hand-drawn sketches. In *Proceedings of the Fifth International Workshop on Frontiers in Handwriting Recognition*, pages 233–238, September 1996.

[16] M. Moyle and A. Cockburn. The design and evaluation of a flick gesture for 'back' and 'forward' in web browsers. In *Proceedings of the Fourth Australasian User Interface Conference (AUIC2003)*, pages 39–46, February 2003.

[17] E. D. Mynatt, T. Igarashi, W. K. Edwards, and A. LaMarca. Flatland: New dimensions in office whiteboards. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 346–353. ACM Press, 1999.

[18] J. P. Pereira, M. J. Fonseca, and J. A. Jorge. Handling ambiguity and errors: Visual languages for calligraphic interaction. In *Proceedings of the 14th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAP'01)*, pages 312–319. IEEE Computer Society, 2001.

[19] D. Rubine. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, pages 329–337. ACM Press, 1991.

[20] G. Russell, Y.-M. Chee, G. Seni, L. Yaeger, C. Tremblay, K. Franke, S. Madhvanath, and M. Froumentin. Ink markup language [online], Available: http://www.w3.org/tr/inkml/ [Accessed 14 October 2003]. *W3C Working Draft*, August 2003.

[21] M. Turk and G. Robertson. Perceptual user interfaces (introduction). *Commun. ACM*, 43(3):32–34, 2000.

[22] G. Viano, A. Parodi, J. Alty, C. Khalil, I. Angulo, D. Biglino, M. Crampes, C. Vaudry, V. Daurensan, and P. Lachaud. Adaptive user interface for process control based on multiagent approach. In *Proceedings of the working conference on Advanced visual interfaces*, pages 201–204. ACM Press, 2000.

[23] A. Wexelblat. Gesture at the user interface: A CHI '95 workshop. *ACM SIGCHI Bulletin*, 28(2):22–26, 1996.

[24] M. J. Woolridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[25] M. J. Woolridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.