JavaSketchIt: Issues in Sketching the Look of User Interfaces

Anabela Caetano and **Neri Goulart** and **Manuel Fonseca** and **Joaquim Jorge**
Department of Information Systems and Computer Science
INESC-ID/IST/Technical University of Lisbon
R. Alves Redol, 9, 1000-029 Lisboa, Portugal
email: atc@mega.ist.utl.pt, nfcg@mega.ist.utl.pt, mjf@inesc-id.pt, jorgej@acm.org

Abstract

We present a visual approach to layout static components of user interfaces as hand-drawn compositions of simple geometric shapes, based on sketch recognition. We have defined a visual grammar using drawing data from target users, where we tried to figure out how people sketch interfaces and what combinations of shapes are more commonly used to define widgets. From these we built our grammar and implemented a prototype, JavaSketchIt, that allows creating user interfaces through hand-drawn geometric shapes, identified by a gesture recognizer. This prototype generates a Java interface, whose layout can be beautified using an *a posteriori* set of grammar rules (*e.g.* to align and group objects). To validate our approach, we conducted usability studies to compare our approach with a commercial system (JBuilder). Besides a measurable speed advantage in drawing interfaces, users found our system more comfortable, natural and intuitive to use, than the competing product, as demonstrated by post–experiment inquiries.

Introduction

The design and coding of the user interface represents a significant percentage of the total time spent in creating applications. Even though interface builders reduce the amount of time needed as compared to manual design, they focus on the final result, rather than allowing users to rapidly explore design ideas (Hearst *et al.* 1998). This emphasis on the final result inhibits the creativity of interface designers, because it suggests false commitment to a particular solution, discouraging users from exploring other alternatives. We believe, as other authors (Landay & Myers 1995; Gross 1996), that better computer–based design tools should support sketching as the primary means to outline and diagram user interfaces.

Since paper and pencil are the designer's choices to quickly sketch new ideas and shapes, we try and approach this environment by proposing a visual method based on composing hand-drawn geometric shapes. In this manner, we exploit designer's natural ability at sketching and drawing.

Our approach combines the usability and flexibility of paper with the plasticity and interactivity of electronic media. To this end, we are exploring a new generation of visual interfaces organized around sketches and visual languages, which we call *Calligraphic Interfaces*.

The rest of the paper is organized as follows. First, we describe related work about sketching user interfaces. The next section addresses fundamental distinctions between sketch–based and direct manipulation applications. Then we describe the task analysis performed to identify the "best" visual grammar. The next section presents the prototype architecture and describes the gesture recognizer, the visual language used to define widgets and the *a posteriori* grammar used to beautify the final result. Finally, we present our experimental evaluation and draw the most relevant conclusions, identifying ongoing and future work.

Related Work

Calligraphic Interfaces predate some of the most established work in Graphical User Interfaces by many years. Indeed, communication through sketches and diagrams precedes the invention of writing by several centuries (Harley & Woodward 1987). In 1963, Sutherland presented the first interactive system, Sketchpad (Sutherland 1963) that used a light pen to draw diagrams directly over the screen surface.

Weitzman (1994) presented an approach to automatic document layout based on parsing and syntax–directed translation through relational grammars. Although their goal was to capture document composition and layout styles through visual grammars they devised an interactive editor to define visual productions and layout rules.

Gross and Do (Gross 1996; Gross & Do 2000) describe a system based on sketches that are partially interpreted for use in architectural drawings, using a stroke recognizer considerably simpler and less robust than ours. His work does not address visual ambiguity also. In general, work in this area does not properly handle the ambiguous nature of visual input. Moreover, when visual ambiguity is addressed the solutions devised are often based on syntax manipulation or resort to imposing arbitrary thresholds on parameters.

Meyer (1996) describes EtchaPad, a tool for sketching widgets in the Pad++ system, including layout operations such as beautifying. Our system tries to accomplish much of this automatically, in the vein of ideas put forth by Pavlidis (1985).

Landay (2001) describes a sketch-based approach (SILK)

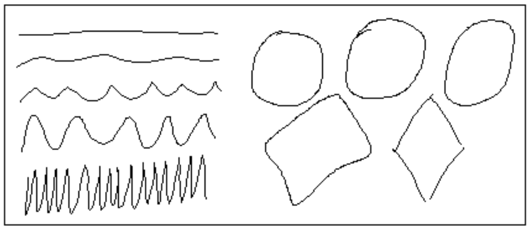

Figure 1: Ambiguity in sketched shapes

for the early stages of user interface design. His work uses electronic sketching to allow designers to quickly record design ideas in a tangible form, obviating the need to specify too much detail too early in the process. His electronic sketches possess the advantages normally associated with computer–based tools, making them well suited to user interface design. While the main focus of our work is the static component of the interface, Landay focuses on the dynamic part, using storyboards to illustrate sequences between screens. SILK keeps the initial sketch visible while we identify the sketch and replace it by a beautified version (but not the final widget) while editing.

Even though Landay's tool supports the design of the static component of the user interface, it is not very clear how the grammar productions are defined for each widget and what parsing approach is used. Also, it is not clear how input errors and misrecognitions are handled, an ever–present issue in recognition–based applications.

Sketching vs Direct Manipulation

Many of the systems reviewed above use a combination of sketch recognition and direct manipulation commands to achieve their results. Our approach is more "radical" in that direct manipulation commands are only used to carry out the most infrequent tasks (opening a file, saving, generating code, etc.). Almost everything else is operated through gestures and sketch recognition.

However, this also poses problems from a parsing visual languages standpoint in that conventional parsing cannot be applied throughout. Indeed, visual grammars require that a start symbol be eventually "recognized" to accept its corresponding visual sentence. This is not adequate for interactive input, in which users construct many different individually valid visual sentences but no coherent whole (visual sentence) until the very last stages of a given design. So, we decided to use what we call *partial goals* or fragments of grammars that can be accepted separately. We address then parsing as a two step phase. To recognize shapes and combine these into widgets and compound widgets we use a *bottom–up* approach. To recognize higher–level constructs (and beautify layouts as side effect) we use a *top–down* separate step. The idea of partial grammars was suggested by Wittenburg and Weitzman (1994).

One of the problems recognizing hand drawn sketches lies in identifying individual shapes. This requires resolving ambiguous visually related shapes. Ambiguity exists, for instance, between `Rectangles` and `Diamonds`, `Ellipses` and `Circles`, `Lines` and `WavyLines`, as Figure 1 shows.

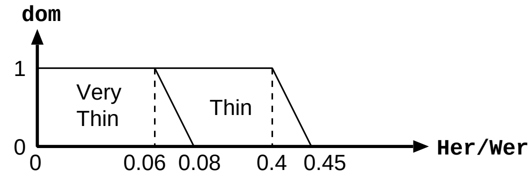

Figure 2: Fuzzy sets representing *Thinness*.

Composing geometric shapes using visual relationships and constraints illustrates another source of ambiguity. Quite often the evaluation of unary predicates that inspect shape properties yields ambiguous results. For example, we use the slope of a line to classify it as vertical, oblique or horizontal.

Humans solve the natural ambiguity associated with individual visual symbols, its properties and its arrangements, by identifying more than one possible characterization and using context or feedback from other people to select one interpretation.

To address these problems we use Fuzzy Relational Grammars. These are described in (Jorge 1994) and in (Albuquerque, Fonseca, & Jorge 2000). Their main advantage is to combine fuzzy logic and spatial relation syntax in a single unified formalism. We address imprecision and ambiguity both at the lexical and syntactical levels. At the lexical level, our recognizer uses fuzzy logic to deal with ambiguous shapes as illustrated in Figure 1. For example the production below describes how to identify `Lines`, regardless of rotation and scale:

```
IF Stroke IS VERY THIN
THEN
    Shape IS Line
```

Fuzzy grammar rules allow us to *quantify* attributes on objects such as `VERY THIN` through *Linguistic Variables* (Zadeh 1965). This particular instance describes the fact that lines are geometrically thin (*i.e.* their enclosing rectangles have a very low Height (Her) to Width (Wer) ratio). Figure 2 illustrates this. Contrarily to crisp logic rules which depend on arbitrary thresholds, linguistic variables allow us to model user and context variations in an elegant and flexible way.

We deal with visual ambiguity in a similar way, *i.e.* when it is not possible to univocally identify a shape, the recognition subsystem returns a list of possible candidates. The application can then choose the "best" candidate using context information.

Finally, at the syntax (grammar) level, fuzzy spatial relations allow us to deal with imprecise spatial combinations of geometric shapes. E.g. the rules below describe productions used to identify the `TextField` and `RadioButton` widgets.

```
IF X-Line INSIDE Rectangle
   AND Rectangle IS THIN
   AND X-Line IS HORIZONTAL
```

```
WHERE X-Line IS Line OR WavyLine
THEN
    Widget IS TextField


IF X-Line RIGHT-OF X-Circle
   AND Circle IS SMALL
   AND X-Line IS HORIZONTAL
WHERE X-Line IS Line OR WavyLine
   AND X-Circle IS Circle OR Ellipse
THEN
    Widget IS RadioButton
```

In the productions above the spatial relation `RIGHT-OF` maps spatial arrangements of objects into fuzzy sets to allow imprecisely drawn arrangements to be flexibly recognized. Figure 3 shows some of these arrangements, some of which are recognized (a, b, c) and some (d, e, f) are not. d) fails, because the `Ellipse` is not small. e) fails, because the `RIGHT-OF` relation does not hold, contrary to c) since the degree of spatial overlap is excessive. Finally, the `Line` in f) is too slanted for the production to fire. Fuzzy logic allows us to percolate degrees of certainty from the right–hand–sides to the left–hand–side of productions and up a parse tree to reflect a "goodness–of–fit" of sketches to grammar entities. Recognition of compound objects happens as a side–effect of certain non–terminals being recognized by the parser. We can handle ambiguity at the user interface level by backtracking on the choice of symbols although this currently only works at the recognizer interface.

Fuzzy Logic and parsing alone cannot guarantee success in addressing users' drawing styles. To identify the "right" visual grammars, we asked likely users of our system, as described in the next section.

Task Analysis

Specifying user interfaces using a paper like interface can be fast, easy and natural. However, to make it intuitive and easy to learn, we must know how users sketch interface widgets. Only then we can define a visual grammar which provides a good match with users' mental models.

We started our task analysis by inquiring ten likely users of our system (undergraduate Comp Sci students) about the best way to represent a widget using a combination of sketches. The inquiry was composed of two parts. In the first, we asked users to sketch a representation for each widget without any restriction. In the second part, we asked the same thing, but now restricted to a set of geometric shapes (see Figure 4). From the responses we selected the two or three most drawn combinations of figures for each widget and we defined our visual grammar, which is depicted in Figure 5.

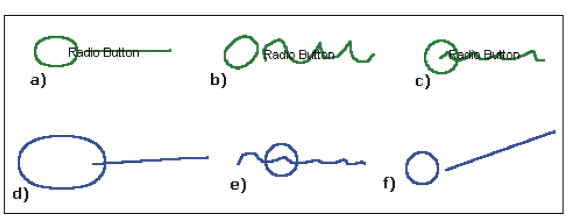

Figure 3: Valid and invalid sketches for `RadioButtons`

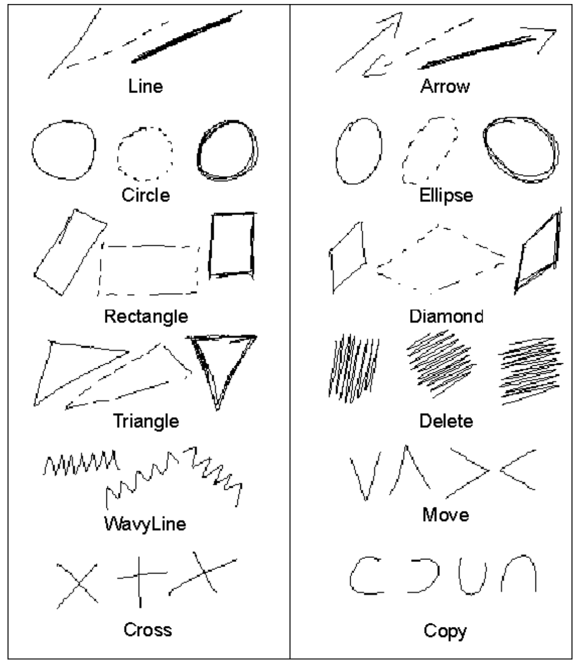

Figure 4: Shapes identified by the recognizer

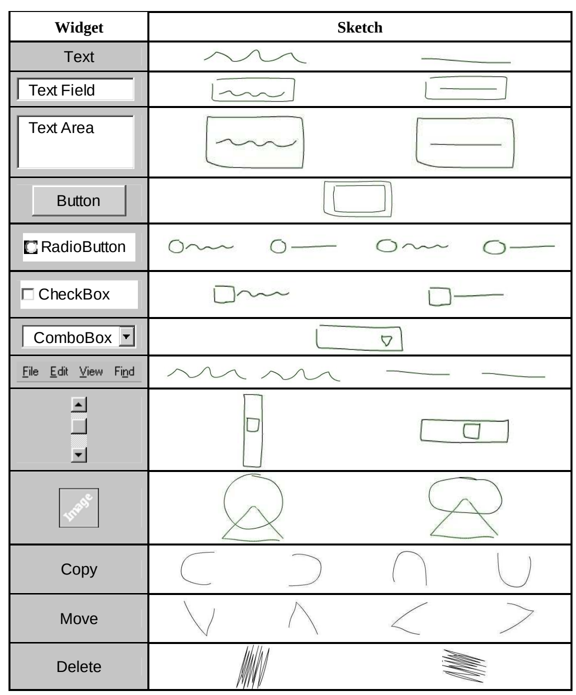

Figure 5: Visual grammar used to define widgets

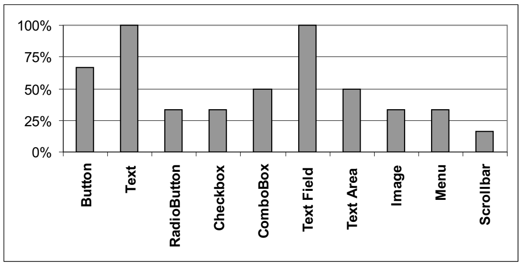

Figure 6: Percentage of combinations made by users similar to our grammar

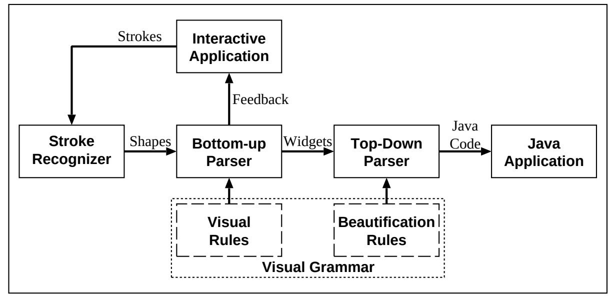

Figure 7: JavaSketchIt architecture

After the implementation of the prototype we wanted to check how intuitive our grammar was. To this end, we made another evaluation with six subjects, who were asked to represent the set of widgets from Figure 5 using just combinations of shapes from Figure 4. Subjects were not aware of which combinations of shapes we had selected for our grammar. After the inquiry we counted how many widgets drawn by users, matched entries in our grammar. The results are shown in percentage in Figure 6.

As we can see, in Figure 6 half of the widgets have more than 50% of combinations similar to ours. Even though some combinations from our grammar were different from users' selections, they found our grammar easy to learn. We consider this result to be encouraging, proving there is a reasonable "user–independent" set of representations of widgets. Another usability test conducted with four users found out that they retained over 90% of the shape combinations after one month not using our system.

System Architecture

Our prototype identifies ten different widgets and supports three edition commands (copy, move and delete).

The interaction sequence starts when the user sketches shapes to compose an interface layout, using a pen on a digitizing tablet. The system tries to identify each shape as soon as it is drawn, using a shape recognizer. After identifying a shape, the system tries to combine it with previously inserted shapes, using basic spatial and adjacency relationships.

This information about shapes and relationships is then fed to a bottom–up parser system, which uses basic knowledge about the structure and patterns of user interface layout to infer which graphic element was intended. The inference rules are grouped into a grammatical specification of a visual language for user interface design. In this visual language symbols are sketches built by combining simple shapes from a subset of those presented in Figure 4. A visual grammar defines rules of composition of individual elements as described in (Jorge 1994). Finally, the user interface as a whole is submitted to a top–down beautification stage, which applies a set of beautification rules to group, align or center widgets.

The system components and their relations are depicted in Figure 7 and described in the following sections.

Stroke Recognizer

The stroke recognizer uses CALI (Fonseca & Jorge 2002), a software library for developing Calligraphic Interfaces. It is a fast, simple and compact approach to identify Scribbles (multi-stroke geometric shapes) drawn with a stylus on a digitizing tablet. The method is able to identify shapes of different sizes and rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines.

The recognizer has a recognition rate of 96%. It is fast: each scribble requires, on average, less than 50 ms (using a Pentium II @ 233 MHz) to be recognized, from feature vector computation to final classification. The fast response characteristic makes it very usable in interactive applications.

Beautification

Besides the composition rules defined for the interface layout, we also have a set of "beautification" rules to be applied *a posteriori*. These rules are defined also as grammar productions but they are parsed top–down rather than bottom–up (as the grammar rules that identify widgets and primitives). We thus employ a mixed strategy, in which top–down and bottom–up phases are employed to parse a visual sentence. According to previous experience, parsing unconstrained recursive productions can lead to exponential re-

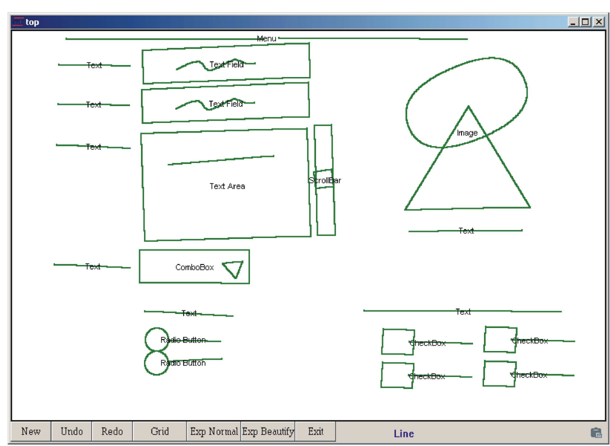

Figure 8: Sketch of an user interface

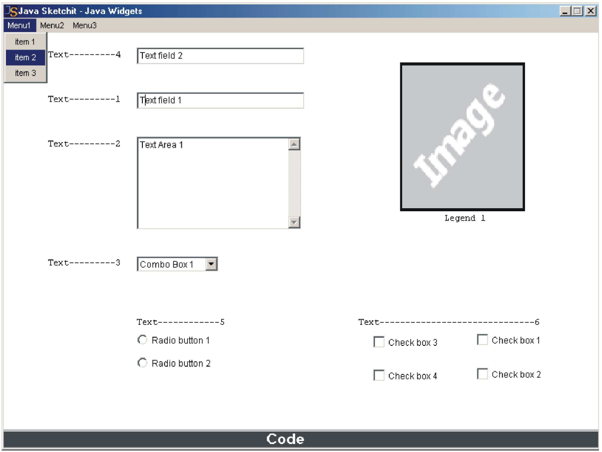

Figure 9: Final user interface

source consumption, thus making it unpractical to use this kind of productions for interactive parsing.

These rules are used to group widgets such as `RadioButtons` or `CheckBoxes` (if they are near each other), to align `Text` and `TextFields` or `TextAreas`, to center a caption below its associated `Image` or to align `Buttons`. Figure 9 shows the final result of an user interface sketched (see Figure 8) using our prototype.

Again these rules exemplify the flexibility and expressive power afforded by Fuzzy Logic and approximate matching in that we can naturally express concepts such as “A and B are approximately aligned” or “B and C are approximately concentric” where A and B are right–hand–side symbols in a grammar production. The beautification occurs as a side–effect of parsing after the left–hand–side is reduced, by *enforcing* the spatial relationships to hold *exactly*, through changing attributes of right–hand–side items. Code generation happens also as a side effect of parse–tree traversal: the portion of the visual sentence, which forms a correct visual sentence (i.e. is reachable from the start symbol) is traversed to produce Java code for each non-terminal node in its corresponding parse–tree.

Experimental Evaluation

We performed an usability study to check the usability of our prototype and to compare it with the JBuilder editor. The study was done by six subjects (3 males and 3 females) using a pen–based computer SONY VAIO LX900, which features

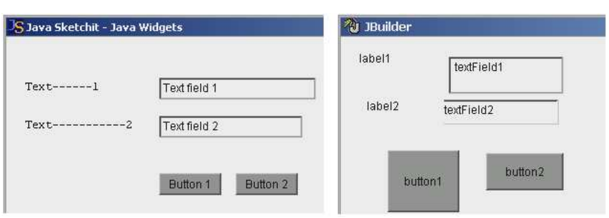

Figure 10: UI built using JavaSketchIt and JBuilder

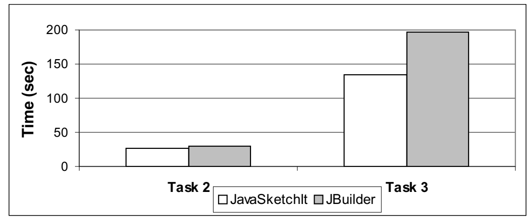

Figure 11: Time spent building a simple(task 2) and a complex (task 3) user interface

an integrated digitizing tablet and display screen.

Our usability study was performed in three steps. First, a preliminary inquiry allowed us to get some information about the users, their experience with pen–based interfaces and how they sketched the different widgets using the set of shapes shown in Figure 4. For the second part, users exercised our system and JBuilder. We provided them with basic instructions about our prototype and about JBuilder. After the training stage, users performed the same tasks using both systems. Finally, we asked them to fill in a post–experiment questionnaire, to get feedback on diverse items such as, satisfaction, preferences, comparative advantages, disadvantages, etc.. During our study we measured the time needed to complete each task, the number of errors and the number of times a user had to consult the manual.

From the first questionnaire we learnt that half of the users were not familiar with pen–based computers and that they usually design user interfaces. Additionally, we confirmed that our representation for widgets is reasonably intuitive, since more than 50% of the sketches spontaneously drawn matched ours (see Figure 6).

The practical part of our usability test revealed that for simple tasks (see Figure 10) our system was as fast as JBuilder, but for more complex interfaces (as the one in Figure 9) users took less time to finish the task with JavaSketchIt. Figure 11 shows the time spent on each task using both applications.

We also did a test to check the ease of learning (*learnability*) of our system. This test consisted in comparing the time and the number of errors from Task 1 and Task 4. The results, as we can see in Figures 12 and 13, show that users took less time and made less errors in the last task. Though, after using our system just for a while, users seemingly had not trouble learning the visual grammar and were able to create user interfaces without too many errors.

Finally, the last questionnaire revealed that on a subjective assessment, users considered our application very easy to use and to learn and that the combination of figures (grammar) used to define widgets is reasonably intuitive. The comparation of both applications revealed that JavaSketchIt is more natural, more intuitive, easier to use and feels more friendly and simpler than its commercial counterpart.

Conclusions and future work

We described a prototype to sketch user interfaces using combinations of simple shapes identified by a gesture recog-

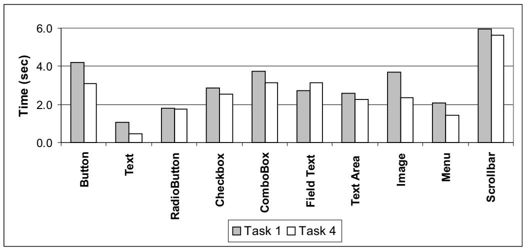

Figure 12: Time spent on the first task and on the last task, showing the learning evolution

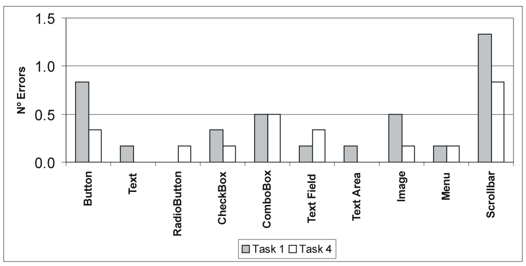

Figure 13: Errors made during the first and the last task

nizer. Before specifying the grammar that defines widgets, we used task analysis to check how users combine shapes to draw widgets. From the results of this study we built our grammar and implemented a prototype. To validate our approach, we made a usability assessment and compared our system with a commercial product, JBuilder. The results shown that our system was very well accepted by users and that they felt comfortable using it. Further, our sketch–based approach was considerably faster than its commercial counterpart for the more complex drawings which we consider to be an encouraging omen for calligraphic interfaces. We plan to expand our system in the future to use richer shape combinations, improve interactive parsing of ambiguous constructs through backtracking at the parser level. Finally we want to extend the beautification approach by a richer set of productions, operators and the possible inclusion of constraint satisfaction mechanisms.

Acknowledgments

This work was funded in part by the Portuguese Foundation for Science and Technology, project 34672/99 and the European Commission, project SmartSketches IST-2000-28169.

References

Albuquerque, M. P.; Fonseca, M. J.; and Jorge, J. A. 2000. Visual Languages for Sketching Documents. In *Proceedings of the IEEE Symposium on Visual Languages (VL'00)*.

Borning, A.; Lin, R.; and Marriott, K. 1997. Constraints for the Web. In *ACM Multimedia 97*. ACM.

Fonseca, M. J., and Jorge, J. A. 2002. CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium*.

Gross, M. D., and Do, E. Y.-L. 2000. Drawing on the back of an envelope: a framework for interacting with application programs by freehand drawing. *Computers & Graphics* 24(6):835–849.

Gross, M. D. 1996. The Electronic Cocktail Napkin - A computational environment for working with design diagrams. *Design Studies* 17(1):53–69.

Harley, J. B., and Woodward, D., eds. 1987. *The History of Cartography*, volume 1. Chicago, IL: University of Chicago Press.

Hearst, M. A.; Gross, M. D.; Landay, J. A.; and Stahovich, T. F. 1998. Sketching Intelligent Systems. *IEEE Intelligent Systems* 13(3):10–19.

Jorge, J. A. 1994. *Parsing Adjacency Grammars for Calligraphic Interfaces*. Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, New York - USA.

Landay, J., and Myers, B. 1995. Interactive Sketching for the Early Stages of User Interface. In *Proceedings of the Conf. on Human Factors in Comp. Syst. (CHI'95)*, 43–50. ACM Press.

Landay, J. A., and Myers, B. A. 2001. Sketching interfaces: Toward more human interface design. *IEEE Computer* 34(3):56–64.

Meyer, J. 1996. Etchapad - disposable sketch based interfaces. In *Proc. ACM SIGCHI'96 - short papers*.

Pavlidis, T., and Wyk, C. J. V. 1985. An automatic beautifier for drawings and illustrations. In *SIGGRAPH'85 Proceedings*, 225–234. ACM.

Sutherland, I. E. 1963. Sketchpad: A Man-Machine Graphical Communication System. In *Spring Joint Computer Conference*, 2–19. AFIPS Press.

Weitzman, L., and Wittenburg, K. 1994. Automatic presentation of multimedia documents using relational grammars. In *ACM Multimedia 94*. ACM Press.

Zadeh, L. A. 1965. Fuzzy sets. *Information and Control* 8:338–353.