A Parsing Technique for Sketch Recognition Systems

Conference Paper · October 2004

DOI: 10.1109/VLHCC.2004.3 · Source: IEEE Xplore

4 authors:

Gennaro Costagliola
Università degli Studi di Salerno
228 PUBLICATIONS **1,974** CITATIONS
SEE PROFILE

Vincenzo Deufemia
Università degli Studi di Salerno
118 PUBLICATIONS **1,053** CITATIONS
SEE PROFILE

Giuseppe Polese
Università degli Studi di Salerno
112 PUBLICATIONS **988** CITATIONS
SEE PROFILE

Michele Risi
Università degli Studi di Salerno
110 PUBLICATIONS **954** CITATIONS
SEE PROFILE

A Parsing Technique for Sketch Recognition Systems

Gennaro Costagliola, Vincenzo Deufemia, Giuseppe Polese and Michele Risi
Dipartimento di Matematica e Informatica
Università di Salerno
84084 Fisciano(SA), Italy
{gcostagliola, deufemia, gpolese, mrisi}@unisa.it

Abstract

Several disciplines require the support of computer-based tools for creating sketches during early design phases. Unfortunately, most computer programs cannot parse and semantically interpret handwritten sketches.
In this paper we present a framework for modeling sketch languages and for generating parsers to recognize them. The underlying parsing technique addresses the issues of stroke clustering and ambiguity resolution in sketches.
We also present a workbench supporting the presented framework.

1. Introduction

Sketches are widely used in engineering and architecture related fields. This is mainly due to the fact that they greatly simplify conceptual design activities through abstract models that let designers express their creativeness, and focus on critical issues rather than on intricate details [13]. Due to their minimalist nature, i.e., representing only what is necessary, they enhance collaboration and communication efficiency. Moreover, the idea of sketch-based user interfaces allows eliminating the redundancy and inefficiency of the early stages of design. In this phase, designers typically spend a considerable amount of time sketching the artifacts through pencil and paper, and only when their main ideas become relatively stable their work is transformed into electronic media.

In recent years we have experienced the development of numerous experimental sketch-based interfaces for a number of different disciplines, including engineering design, user interface design and architectures [3, 7, 15, 18]. Researchers have shown that such applications can combine many of the benefits of paper-based sketching with current electronic tools, enabling new creative and collaborative usage scenarios [14].

The first challenge in building a sketch-based system is recognizing the meaningful patterns implied by a user's pen stroke. This task is not trivial because pattern matching must be flexible enough to allow some tolerance in sketch recognition, but sufficiently constrained not to accept incorrect patterns. Moreover, free hand sketching is an inherently imprecise process and varies greatly from person to person. As an example, a user-drawn square might easily be misinterpreted as a circle if it is drawn with rounded corners, or even a circle might be misinterpreted as a sloppily-drawn square. Furthermore, semantically different objects might be graphically represented by identical or apparently similar symbols.

Without intelligent treatment of this and other forms of drawing ambiguity, accurate sketch recognition is not possible. Consequently, sketch recognition solely based on shape has a number of serious drawbacks. In fact, recognition algorithms such as Rubine's [19] recognize shapes only from individual strokes, which alone do not convey important contextual information. However, the context in which a particular stroke or group of strokes appears considerably influences the interpretation of that stroke. Hence, recognition involves both the identification of patterns and the interpretation of those patterns with respect to the context in which they appear. Context plays a pivotal role in resolving ambiguities. From a visual language point of view, this means that the interpretation of a graphical object is strongly influenced by the objects surrounding it.

Another important issue in sketch understanding concerns *ink parsing*, which refers to the task of grouping and separating the user's strokes into clusters of intended symbols, without requiring the user indicate when one symbol ends and the next one begins. Technically, ink parsing tries to identify distinct symbols from a continuous stream of pen strokes.

In this paper we propose a recognition framework for modeling sketch languages and generating the server parser for their recognition. The sketch recognition technique is an extension of LR based parsing techniques, and includes ink parsing and context disambiguation.

Once integrated into a sketch editor, the parser incrementally and unobtrusively interprets the sketches as they are created. Thus, the proposed approach is especially tailored to interactive recognition environments, because the system is aware at any point of possible future interpretations of an incomplete

Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing (VLHCC'04)
0-7803-8696-5/04 $ 20.00 IEEE

sketch. Moreover, the use of well known visual language parsing methodologies allows us to benefit from many of the results reached in this field, such as flexible semantic interpretation and code generation.

The paper is organized as follows. We first discuss the related work in Section 2. In Section 3 we describe our approach for the resolution of main challenges in the creation of sketch recognition frameworks. In Section 4 we detail the formalism for describing sketch languages, and the parsing approach underlying the proposed framework. We then present *SketchBench*, a system for the generation of parsers capable of recognizing hand-sketched sentences of a language. Finally, the conclusion and further research are discussed in Section 6.

2. Related Work

Starting with Sutherland's sketchpad [23], there has been a considerable amount of work devoted to the creation of tools processing people's freehand drawings. Advances in machine intelligence, hardware technology, and computer graphics have accelerated this transition.

The toolkits to support sketch-based application prototyping provide facilities for interpreting and beautifying sketched ink [10], and for user correction of incorrect interpretations [17]. Nevertheless, the problem of robust sketch recognition has been largely ignored and is crucial to the success of sketch-based user interfaces in the real world.

The sketch recognition systems described in [8, 9, 12, 22] use variants of visual language grammars to provide contextual information in order to face the inherent ambiguity of sketched drawings. However, although these early efforts provide a context for resolving ambiguity, their use of grammars was not focused on ambiguity but to find a concise way to specify valid visual structures.

SILK is an interactive sketching tool allowing designers to quickly sketch out a user interface and transform it into a fully operational system [13]. As the designer draws sketches, the recognizer of SILK matches the pen strokes against symbols representing various user interface components, and returns the most likely interpretation. Their recognizer is limited to single-stroke shapes drawn in certain preferred orientations. SILK also addresses the notion of ambiguity in the sketch, but is limited to resolving ambiguity of single parts. This means that once it resolves an ambiguity on a given group of strokes, its decision does not affect the rest of the recognition process.

The Electronic Cocktail Napkin [8] is a sketch-based conceptual design tool that serves as an interface for knowledge-based critiquing, simulation and information retrieval. The system employs a bottom-up recognition method that also includes a method for representing ambiguities in the user's sketches, and is capable of refining its early interpretations of user's strokes by analyzing the surrounding context. In order for ECN to resolve ambiguity, the user must either explicitly inform the system of the correct interpretation, or the system must find a specific higher-level pattern that would provide the context to disambiguate the stroke interpretation.

Starting from a survey on existing recognizers, Mankoff *et. al.* [16] have presented a set of ambiguity resolution strategies, which are concerned with how ambiguity should be presented to the user and how the user should indicate his or her intention to the software. This work highlights a number of critical issues that demand consideration for a better interaction strategy between the end user and the software.

ASSIST is a system that can interpret and simulate a variety of simple hand-drawn mechanical systems [1]. ASSIST presents a formalism for augmenting implicit and explicit feedbacks from the user with contextual information to disambiguate multiple interpretations of a drawing. The main limitation is that its shape recognizers can only recognize a limited number of shapes specifically designed for the particular domain since they are hand-coded. Nevertheless, this work addresses a number of important issues in sketch-based interface design, including user involvement in recognition, ambiguity resolution, and the level of software aggressiveness in recognition.

A statistical visual language model for ink parsing has been introduced in [21]. The approach combines a visual language formalism with a statistical model for disambiguation. Thus, the construction of a new recognizer consists of writing a declarative context-free grammar for the language, generating a model from the grammar, and training the model on drawing examples. The main limitation of this approach is that it is designed to parse the component elements of a composite symbol, rather than parsing entire sketches. In fact, although theoretically possible, this method is not suitable for parsing large scale sketches. This is because their training algorithm requires a large amount of training examples to learn each of the parse models. Moreover, the learned models do not generalize to new sketches that were not seen during training. Therefore, to be able to parse entire sketches, one would need to train the system on every plausible sketch that can be created, which is clearly impossible.

An architecture supporting the development of robust recognition systems across multiple domains has been introduced in [2]. The architecture maintains a separation between low-level shape information and high-level domain-specific context information, but it uses the two sources of information together to improve recognition accuracy. Moreover, in order to disambiguate between multiple interpretations of a sketch the approach allows high-level interpretations to guide low-level recognition accuracy.

Sim-U-Sketch is a sketch-based interface for Simulink software package [11]. The system includes a domain-independent, multi-stroke, trainable symbol

recognizer having the advantage of learning new definitions from single prototype examples. Moreover, it includes a multi-level parsing scheme that allows users to continuously sketch. The parser uses contextual knowledge to both improve accuracy and reduce recognition times. However, the recognition process is guided from "marker symbols", which are symbols easy to recognize, that they assume to exist always in the sketch.

3. Stroke Clustering and Ambiguity Resolution

Ink parsing and ambiguity resolution are the two main challenges in the creation of a sketch recognition framework that is applicable to several application domains. In this section we present our approach to solve these problems.

First, we have to determine a set of related clusters of strokes, each of which could potentially represent one of the language symbols. To this end, we propose incremental visual language parsers. In particular, we represent domain-specific shapes using extended positional grammars [5]. Such grammars allow us to define composite objects by relating simple objects through spatial relations. In the sketch recognition domain they allow us to specify shapes as compositions of multi-stroke symbols. However, as opposed to icon languages where the symbols are provided in palettes, hand-sketched shapes are imprecise (i.e., contain ambiguity). As an example, Figure 3.1(a) shows a hand-sketched class symbol of a UML class diagram, where some strokes are inaccurately drawn, and are not strictly connected. Such types of ambiguities have been classified by Furtrelle as structural ambiguities, since they involve spatial relationships between objects [6].

In order to take into account these ambiguities in the grammar specification, we introduce tolerances in the relations between the symbols. These are realized through thresholds allowing us to make the satisfiability of the relations between strokes more flexible.

In the literature there are two main approaches to symbol recognition: hierarchical and image-based. The former sees strokes as compositions of primitive strokes, such as lines and curves, whereas the latter exploits image approximate match paradigms to map groups of strokes onto language symbols. One of the advantages of image-based approaches is that they do not require the specification of primitive strokes and corresponding composition rules. Moreover, they are more suitable for recognizing complex graphical symbols. On the other hand, hierarchical approaches are more suitable and efficient for recognizing languages whose symbols are composed of few primitive strokes. Thus, they are suitable for recognizing diagrammatic languages.

Our approach naturally supports hierarchical stroke recognition thanks to the underlying grammar formalism, which simplifies the specification of compositional sketched symbols.

The mapping of initial clusters into language symbols provides the recognizer with a growing context that drives the recognition of the remaining part of the input sketch. In particular, our framework uses spatial context to aid recognition. This provides information about the shape that is more likely to occur in a given context for the given domain. For example, in UML diagrams, "association" relations (represented with lines) often connect classes. Thus, if a class is recognized, and a line sketch connected to it is mapped to an association, then the recognizer expects a class sketch be connected to the other line end, as shown in Figure 3.1(b).

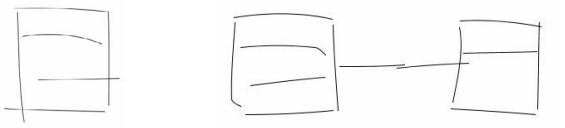

Figure 3.1. Structural ambiguity in a shape (a) and in a sketch (b)

Another important problem in symbol recognition is that of multiple interpretations of a sketch. To this end, we have used a nondeterministic parser that assigns a rank value to each candidate interpretation of a sketch or part of it. Such rank value is computed by combining the accuracy of the strokes forming the sketch and of their spatial relations. In particular, the first value is computed by using a *shape recognizer* (currently implemented by utilizing the recognizer distributed as a part of SATIN [10]), which returns a rank list of primitive shapes similar to the input stroke, whereas the second value is computed by evaluating the spatial relations occurring between sketched strokes. More formally, let w be a sketch and P be the set of relations applied for its recognition, the interpretation value $I(w)$ is given by:

$$I(w) = \frac{1}{|w|}(k_s * \sum_{s \in w} sim(s)) + \frac{1}{|P|}(k_r * \sum_{r \in P} acc(r))$$

where $k_s + k_r = 1$, $sim(s)$ is the similarity value returned by the shape recognizer on the stroke $s \in w$ with respect to the primitive shapes, $acc(r)$ is the accuracy of the relation r when the parser applies it between two strokes in w, and k_s and k_r are weights which allow us to configure the recognizer in order to make it more stroke or relation oriented. The choice of the right balance between these two weights can be based on the level of expertise of the user. As an example, a user familiar with the language will sketch strokes quickly. Thus, s/he is more likely to produce imprecise but suitably related strokes, which should suggest us to overweight k_s with respect to k_r.

4. LR-based Sketch Parsing

In the next subsection, we illustrate how to model sketching languages with the formalism of *eXtended Positional Grammars* (XPG, for short). Successively, we present an incremental parser for sketch recognition.

4.1. A Grammar Formalism for Modeling Sketch Diagrams

XPGs represent a direct extension of context-free string grammars, where more general relations other than concatenation are allowed. The idea behind the definition of such formalism has been to overcome the inefficiency of visual languages parsing algorithms by searching suitable extensions of the well-known LR technique. In particular, the Extended Positional Grammars formalism is based on an extension of LR parsing named XpLR methodology [5].

The XPG formalism conceives a sentence as a set of symbols with attributes. Such attributes can be classified in physical, syntactic, and semantic attributes. The physical component represents the features of a symbol and allows us to materialize the sentence to our senses; whereas the values of the syntactic attributes are determined by the relationships holding among the symbols. Thus, a sentence is specified by combining symbols with relations. The values of semantic attributes specify the semantic interpretation of a symbol, and they are used after parsing to perform semantic checks and translation tasks.

As a consequence, to model sketch languages with XPGs it is necessary to introduce a further step of specification, in which the symbols of the language are defined as a set of strokes with attributes. Since the number of possible strokes is infinite, the symbols are specified by combining elementary shapes (such as lines, circles, curves, etc.) through relations. In this case the relations depend on the physical attributes of the shapes.

As an example, the states in state transition diagrams could be defined as an elementary circle, or as a set of arcs describing a circumference. Moreover, the syntactic attribute to express the attachment relation between the borderline of a state and the start or end points of edges can be represented by an "attaching region" on that state. Successively, a state transition diagram could be specified by using the defined symbols representing states and edges, and the relations between them.

More formally, an Extended Positional Grammar is the pair (G, PE), where PE is a *positional evaluator*, and G is a particular type of context-free string attributed grammar (N, T∪POS, S, P) where:

- N is a finite non-empty set of *non-terminal* symbols;
- T is a finite non-empty set of *terminal* symbols, with $N \cap T = \emptyset$;
- POS is a finite set of *binary relation* identifiers, with $POS \cap N = \emptyset$ and $POS \cap T = \emptyset$;
- $S \in N$ denotes the starting symbol;
- P is a finite non-empty set of *productions* having the following format:

$$A \rightarrow x_1 R_1 x_2 R_2 \ldots x_{m-1} R_{m-1} x_m \ \Delta, \Gamma$$

where A is a non-terminal symbol, $x_1R_1x_2R_2\ldots x_{m-1}R_{m-1}$ x_m is a linear representation with respect to POS where each x_i is a symbol in N∪T and each R_j is partitioned in two sub-sequences $(\langle REL_{j_1}{}^{h_1}(t_1),..,REL_{j_k}{}^{h_k}(t_k)\rangle$, $\langle REL_{j_{k+1}}{}^{h_{k+1}}(t_{k+1}),...,REL_{j_n}{}^{h_n}(t_n)\rangle)$ with $1 \le k \le n$.
Each $REL_{j_i}{}^{h_i}(t_i)$ relates physical or syntactic attributes of x_{j+1} with physical or syntactic attributes of x_{j-h_i}, with $0 \le h_i < j$, by means of a threshold t_i. Notice that we denote $REL_1{}^0(0)$ simply as REL_1. The relation identifiers in the first sub-sequence of an R_j, named *driver relations*, are used during syntax analysis to determine the next symbol to be scanned; whereas the ones in the second sub-sequence, named *tester relations*, are used to check whether the last scanned symbol (terminal or non-terminal) is properly related to previously scanned symbols. We refer to *driver*(R_j) (*tester*(R_j), resp.) as the driver (tester, resp.) relations of R_j.
Δ is a set of rules used to synthesize the values of the syntactic/physical attributes of A from those of x_1, x_2 ,…, x_m;
Γ is a set of triples $\{(N_j, Cond_j, \Delta_j)\}_{j=1\ldots t}$, $t \ge 0$, used to dynamically insert new symbols in the input visual sentence during the parsing process. In particular,

- N_j is a terminal symbol to be inserted in the input visual sentence;
- $Cond_j$ is a pre-condition to be verified in order to insert N_j;
- Δ_j is the rule used to compute the values of the syntactic attributes of N_j from those of x_1,.., x_m.

Informally, a Positional Evaluator PE is a materialization function that transforms a linear representation into the corresponding visual sentence in the graphical representation.

The *language described by an XPG*, *L*(*XPG*), is the set of the visual sentences from the starting symbol S of XPG [5].

An XPG for modeling a sketch language L can be logically partitioned into two XPG grammars. The first, named *ink grammar*, defines the symbols of the language L as geometric compositions of primitive shapes. Thus, the terminals of such grammar are the patterns that cannot be recognized as a combination of other shapes, and must be recognized directly.

As an example, for the state transition diagram language the final state symbol is defined as two concentric circles, as described by the following production:

FState → Circle′ <*concentric*(*0.5*)> Circle″
Δ: $(FState_{attach}$=Circle′$_{borderline})$

where Circle is a nonterminal defining a circle as formed by a single stroke shape or by a multi-stroke shape.

The relations used in these productions are geometric relations, which work on the physical attributes of the symbols. The Δ rules allow us calculating the syntactic/physical attributes of the composite symbols. In the previous example, the syntactic attribute *attach* of *FState* corresponds to the physical attribute *borderline* of *Circle*.

The second grammar, named *language grammar*, specifies the sentences of the language as compositions of shapes defined in the ink grammar through spatial relations. Thus, the terminals of such grammar correspond to the nonterminals of the ink grammar. As an example, the following production uses the *FState* nonterminal previously defined:

$\text{Graph} \rightarrow \text{Graph}' <Link_{1,1}(0,8)> \text{Edge} <Link_{2,attach}(0,8)> \text{FState}$
Δ: $(\text{Graph}_1 = \text{Graph}'_1 - \text{Edge}_1)$,
Γ: $\{(\text{PLACEHOLD}; |\text{FState}_{attach}|>1; \text{PLACEHOLD}_{attach} = \text{FState}_{attach} - \text{Edge}_2)\}$

The relations used in these productions are spatial relations, which work on the syntactic attributes of the symbols computed in the Δ rules [5].

For many relations it is possible to specify semantically inverse relations. In particular, let REL_1 and REL_2 be two relation identifiers, if x $REL_1(t)$ y and y $REL_2(t)$ x hold for the same pairs of symbols x and y, and the same thresholds t, then REL_2 is the *inverse relation* of REL_1, and vice versa. In the following, we denote with *inv*(R) the inverse relation of R. Given an XPG, if there exists *inv*(R) for each R in POS, then XPG is named *reversible*. Now we are ready to introduce the concept of reverse grammar for XPGs that will be useful for the parsing algorithm.

A *reverse grammar* with respect to a reversible XPG G=((N, T∪POS, S, P), PE), denoted with *rev*(G), is an XPG G′=((N, T∪POS′, S, P′), PE), where POS′=*inv*(POS) and P′ is defined as in [4].

Note that L(G) is equivalent to L(*rev*(G)) for each reversible XPG grammar G.

4.2 An LR-based Parser for Sketch Diagrams

A peculiarity of the parsing methodology based on XPG, named *XpLR parsing*, is its scanning of the input in a non-sequential way (i.e., driven by the relations used in the grammar). However, this characteristic increases the occurrences of parsing conflicts, and makes it difficult to establish the symbol of a sentence from which the parsing process has to start.

To solve these problems the XpLR parser has been extended with functionality of non-determinism, incrementality, and bi-directionality [4]. The use of non-determinism in the parsing (namely generalized parsing) eliminates the need for most "grammar-hacking", and allows a syntax specification that naturally corresponds to abstract syntax. The incrementality is fundamental for the construction of visual interactive environments, since it allows us to provide immediate feedback while the user composes a sentence. Finally, the use of two parsers that proceed simultaneously from the same symbol allows scanning the input sentence in opposite directions from an arbitrary starting symbol.

These characteristics allow us to use this type of parser as a sketch recognizer.

Figure 4.1 shows the architecture of the proposed framework. As the strokes are sketched into the editor, a pattern recognizer tries to match the stroke with the elementary shapes (registered in a pattern database). The recognizer produces the result of the pattern matching as a list of classifications and probabilities ordered by probability.

The input sketched symbols together with the information provided by the pattern recognizer are stored in a dictionary.

Given an XPG G for a sketch language L, the G-XpLR parsing tables for G and for its reverse version *rev*(G) are created. Then, for each state in G (*rev*(G), resp.) reachable after the occurrence of the starting symbol the algorithm starts an incremental parser, named *forward* (*backward*, resp.). The forward parsers interact with the backward parsers only when a parser tries to reduce a production. In this case, that parser waits for a *rendezvous*, i.e., an opposite parser attempting to apply the reverse version of the same reduction. The forward and backward parser stacks can be considered as only one graph stack expanding to the right and to the left, and with two types of nodes: *simple stack node* and *joint stack node*. The latter encloses a bipartite graph whose elements are simple stack nodes from forward and backward parsers.

The input to the bi-directional incremental parser is the dictionary storing the information of the modified sketched sentence as produced by the pattern recognizer, a parse forest and a graph stack built on the original sketched sentence.

The parser retrieves the objects in the dictionary by a *Fetch_Stroke* operation driven by the relations in the grammar. The parser implicitly builds and parses a linear representation from the input representation.

The output of the parser is a *probabilistic parse forest* which describes the possible interpretations of the input sketch. Each node of a tree has associated a probability representing the accuracy of the stroke interpretation associated to the leaves of its subtree. In particular, such value is computed during the reduction of the applied productions by using the interpretation value formula introduced in the previous section. Thus, the parser incrementally matches the modified sketch sentence with the leaves of the parse forest, restructures the probabilistic parse forest on the base of the modifications, and updates the graph stack.

Each tree in the parse forest corresponds to an interpretation of the sketch sentence. Such trees can be analyzed to obtain a rank of the interpretations by considering the probability associated to the roots of the trees and the number of language symbols recognized, similarly to what done for natural languages [24]. The

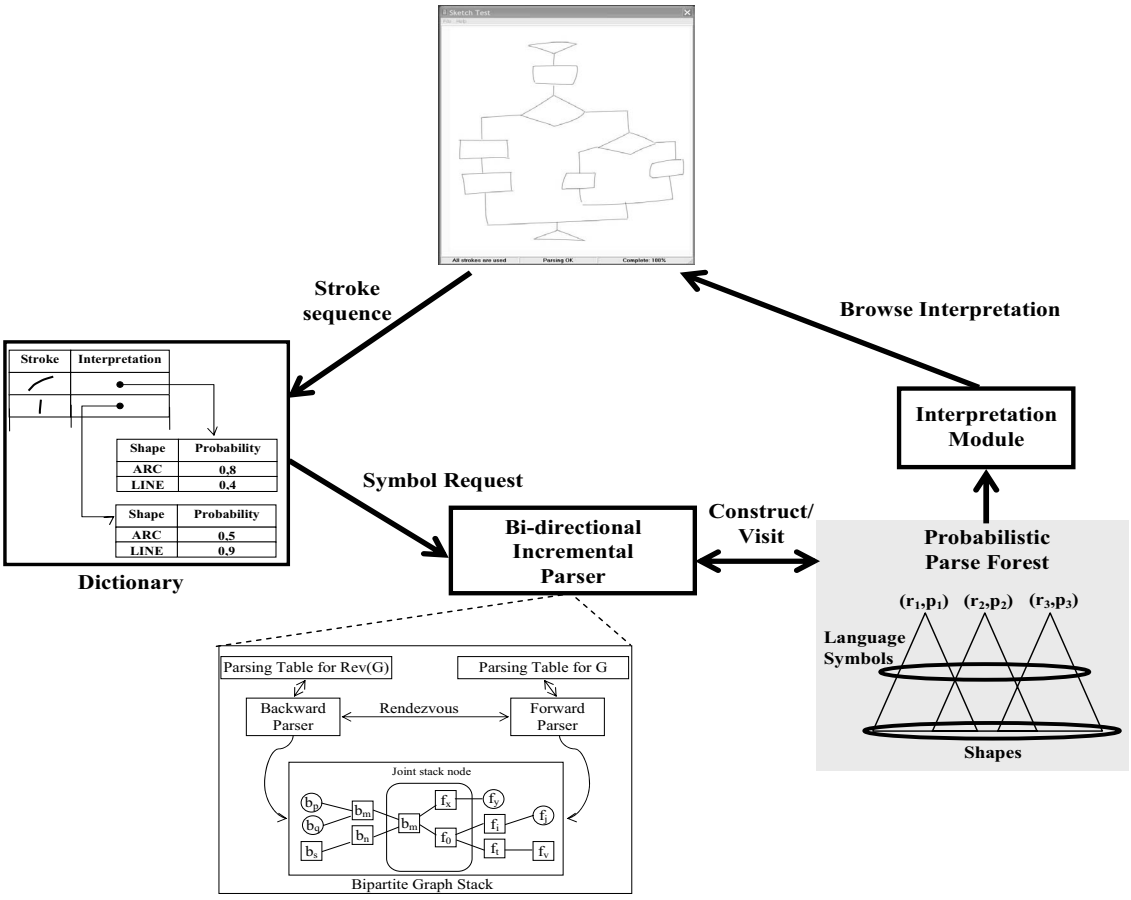

Figure 4.1. The architecture of the proposed framework

editor uses such rank together with the parse forest to show the groups of strokes composing a sentence of the language, so that the user can choose the intended interpretation.

5. SketchBench

This approach has been integrated into the *SketchBench* system, which inherits and extends to the sketch recognition field, concepts and techniques of visual compiler generation tools based on YACC. The architecture of SketchBench is shown in Figure 5.1. SketchBench assists the designer in the specification of the grammar. In particular, the ink grammar can be visually specified through the Symbol Editor. The latter enables the definition of the language symbols and their automatic translation into the grammar productions (see Figure 5.2).

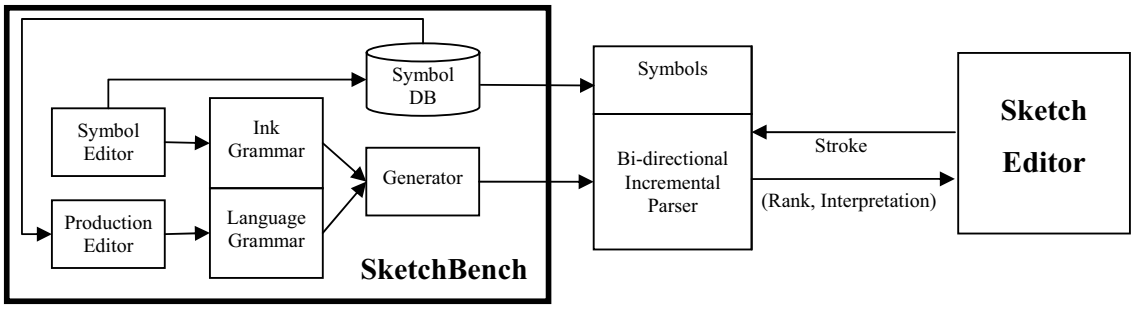

Figure 5.1. The SketchBench architecture

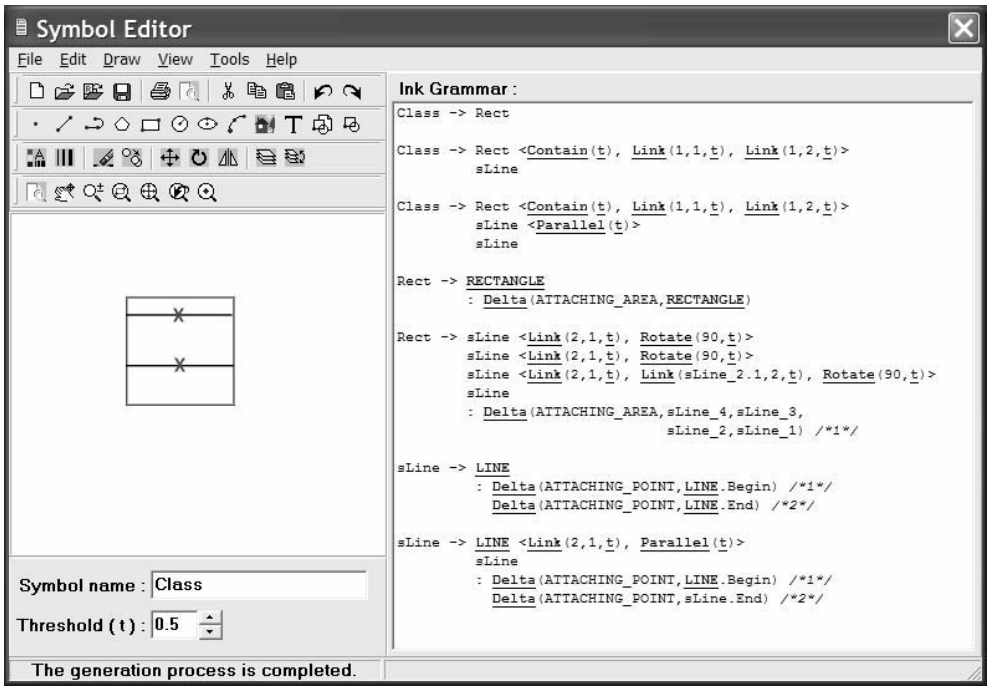

Figure 5.2. Ink grammar definition with SketchBench

By combining the productions generated by the Symbol Editor and the Production Editor, we obtain the complete specification of the XPG grammar.

Starting from such specification, SketchBench automatically constructs its reverse version, and generates a bi-directional parser for the recognition of hand-sketched sentences of the language. Moreover, the set of symbols used in the specification of the ink grammar are included as components of the generated parser.

A generic sketch editor is able to interact with the generated parser, which provides the candidate interpretations of the sketched strokes. In particular, when the sketch is modified, the visual editor informs the parser about the changes. Therefore, the parser first invokes the pattern recognizer of SATIN [10], a Java library supporting handwriting recognition functionalities, and then applies incremental parsing to analyze the attribute-based representation of the sketch, in order to produce the set of its possible interpretations. The interpretations are couples (*rank value*, *interpretated sentence*), which can be visualized in a particular window of the sketch editor. The symbols used in this window are those specified in the Symbol Editor at parser specification time. As an example, Figure 5.3 shows the recognition of a class diagram by using the sketch editor and the parser generated with SketchBench.

Once the system interprets a user's sketch, it can trigger the general semantic actions that have been specified by the user.

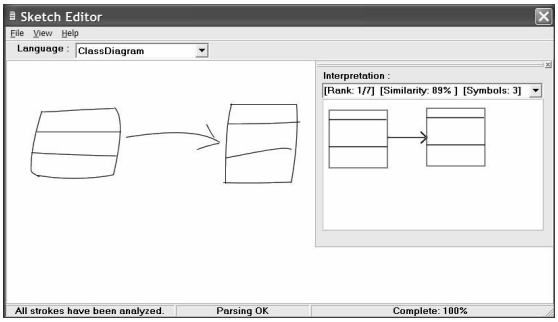

Figure 5.3. Sketch interpretation in SketchBench

6. Conclusions and Further Research

In this paper we have presented a framework for modeling sketch languages and for generating parsers recognizing them. The approach relies on an LR based parsing technique, which allows us to solve both the problem of stroke clustering (i.e., ink parsing) and that of context based ambiguity resolution.

We have also presented the SketchBench system, a tool supporting the presented framework from the early phases of sketch language modeling, such as shape modeling and grammar specification, to the generation of the final parser. The parser is then integrated into a sketch editor, where sketches are incrementally and unobtrusively interpreted as they are drawn. The use of well known visual language parsing methodologies and

powerful grammar formalisms enables the application of the approach to several application domains. In particular, the approach is well suited to applications requiring the parsing of user's strokes in real time, without limiting the designer's drawing freedom. Thus, such parser will enable the creation of powerful and natural early-stage computer aided design tools. Moreover, many of the results achieved in the visual language field, such as flexible semantic interpretation and code generation can be inherited.

In the future we aim to use island parsing [20] to enhance performances and accuracy of the recognition process. In particular, we plan to develop a recognizer that first scans parts of sketches that are easy to recognize by means of the bottom-up parsing technique proposed in this paper. Then, a top-down approach could be used to enforce an interpretation on the ambiguous parts, with the possibility of using Bayesian networks to evaluate missing data or noise in sketches.

Moreover, we also plan to include temporal context in the disambiguation process. Such context provides information about the order in which components are likely to be drawn. For example, in UML class diagrams, classes are usually drawn before the associations connecting them.

We are currently conducting usability studies to test our workbench.

References

[1] C. Alvarado and R. Davis, "Resolving Ambiguities to Create a Natural Sketch based Interface", in *Proceedings IJCAI'01*, Seattle, Vol. 2, 2001, pp. 1365-1371.

[2] C. Alvarado, M. Oltmans, R. Davis, (2002), "A Framework for Multi-domain Sketch Recognition", in *Proc. AAAI Spring Symposium on Sketch Understanding*, 2002, pp. 1-8.

[3] D. Blostein and L. Haken, "Using Diagram Generation Software to Improve Diagram Recognition: A Case Study of Music Notation", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21(11), 1999, pp. 1121-1136.

[4] G. Costagliola and V. Deufemia, "Visual Language Editors based on LR Parsing Techniques", in *Proceeding of SIGPARSE/ACL 8th International Workshop in Parsing Technologies*, Nancy, France, 2003, pp. 79-90.

[5] G. Costagliola and G. Polese, "Extended Positional Grammars", in *Proceedings of 2000 IEEE Symposium on Visual Languages*, Seattle, WA, USA, September 10-13, 2000, pp. 103-110.

[6] R. Futrelle, "Ambiguity in Visual Language Theory and its Role in Diagram Parsing", in *Proceedings of* 1999 *IEEE Symposium on Visual Languages*, Tokyo, JP, 1999, pp. 172-175.

[7] M.D. Gross, "Stretch-A-Sketch: a Dynamic Diagrammer", in *Procs. IEEE Symposium on Visual Languages*, Los Alamitos, CA, USA, 1994, pp. 232-238.

[8] M.D. Gross, "The Electronic Cocktail Napkin – A Computational Environment for Working with Design Diagrams", *Design Studies* 17(1), pp. 53-69, 1996.

[9] R. Helm, K. Marriott, and M. Odersky, "Building Visual Language Parsers", in *Proceedings of CHI '91*, New Orleans, LA, April 1991, pp. 105-112.

[10] J. I. Hong and J. A. Landay. "SATIN: a Toolkit for Informal Ink-based Applications", in *Proceedings of the 13th ACM symposium on User interface software and technology*, pp. 63-72, ACM Press, Nov. 2000.

[11] L.B. Kara and T.F. Stahovich, "Sim-U-Sketch: a Sketch-Based Interface for Simulink", in *Proceedings of Advanced Visual Interface (AVI'04)*, Gallipoli, Italy, 25-28 May 2004, pp. 354-357, ACM Press.

[12] J. Landay, "Interactive Sketching for the Early Stages of User Interface Design", Ph.D. dissertation, Report #CMU-CS-96-201, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1996.

[13] J. Landay and B. Myers, "Sketching interfaces: Toward more human interface design", *IEEE Computer* 34(3), pp. 56-64, 2001.

[14] J. Landay and B. Myers, "Interactive Sketching for the Early Stages of User Interface Design", in *Proceedings of CHI '95*, Denver, CO, 1995, pp. 43-50.

[15] J. Lin, M. Newman, J. Hong, and J. Landay, "DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design", in *CHI Letters: Human Factors in Computing Systems*, CHI 2000, 2(1), pp. 510-517.

[16] J. Mankoff, G. Abowd, and S. Hudson, "Oops: a Toolkit Supporting Mediation Techniques for Resolving Ambiguity in Recognition-based Interfaces", *Computers and Graphics* 24(6), pp. 819-834, 2000.

[17] J. Mankoff, S. Hudson, and G. Abowd, "Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces", in *Proceedings of CHI 2000*, April, 2000, pp. 77-78.

[18] N. Matsakis, "Recognition of Handwritten Mathematical Expressions", Master's Report, Massachusetts Institute of Technology, 1999.

[19] D. Rubine, "Specifying Gestures by Example", *Computer Graphics* 25, pp. 329-337, 1991.

[20] O. Stock, R. Falcone, P. Insinnamo, "Bidirectional Charts: A Potential Technique for Parsing Spoken Natural Language", *Computer Speech and Language* 3(3), pp. 219-237, 1989.

[21] M. Shilman, H. Pasula, S. Russell, and R. Newton, "Statistical Visual Language Models for Ink Parsing", in *Proc. AAAI Spring Symposium on Sketch Understanding*, Stanford, March 2002, pp. 126-132.

[22] B. Shizuki, H. Yamada, K. Iizuka, J. Tanala, "A Unified Approach for Interpreting Handwritten Strokes", in *Procs. 2003 IEEE Symposium on Human-Centric Computing (HCC03)*, Auckland, New Zealand, October 28-31, 2003, pp. 180-182.

[23] I.E. Sutherland, "Sketchpad — A Man-machine Graphical Communication System". PhD thesis, MIT, 1963.

[24] M. Tomita, "Efficient Parsing for Natural Languages - A Fast Algorithm for Practical Systems", Kluwer Academic Publishers, 1986.