Representing and Parsing Sketched Symbols Using Adjacency Grammars and a Grid-Directed Parser

Joan Mas[1], Joaquim A. Jorge[2], Gemma Sanchez[1], and Josep Llados[1]

[1] Computer Vision Center, Computer Science Department, Edifici O Campus UAB, Bellaterra, Spain
{jmas,gemma,josep}@cvc.uab.es
[2] Departamento de Engenharia Informática, INESC, Rua Alves Redol, 9, Lisboa, Portugal
jaj@rtr.inesc-id.pt

Abstract. While much work has been done in Structural and Syntactical Pattern Recognition applied to drawings, most approaches are non-interactive. However, the recent emergence of viable pen-computers makes it desirable to handle pen-input such as sketches and drawings interactively. This paper presents a syntax-directed approach to parse sketches based on Relational Adjacency Grammars, which describe spatial and topological relations among parts of a sketch. Our approach uses a 2D grid to avoid re-scanning all the previous input whenever new strokes entered into the system, thus speeding up parsing considerably. To evaluate the performance of our approach we have tested the system using non-trivial inputs analyzed with two different grammars, one to design user interfaces and the other to describe floor-plans. The results clearly show the effectiveness of our approach and demonstrate good scalability to larger drawings.

1 Introduction

Sketching interfaces are a useful and natural way for people to communicate with computers. By using a digital pen, users can input information such as cursive script annotations or, in a graphical domain, freehand diagrams or graphical gestures. Sketch recognition is therefore a powerful tool in disciplines such as architecture or engineering. In the graphical domain, sketches have an important value. Indeed, sketches provide the ability of expressing complex ideas with simple visual notations, and are a fluent way of human-computer interaction. From a technical point of view, according to Liu's [4] interesting survey, on-line graphics recognition processes may be divided into three main parts: Primitive Shape Detection, Composite Object Recognition and Sketch Understanding. In this work we focus on symbol recognition in sketching diagrams.

Sketches are collections of strokes, i.e. line drawings where basic primitives are the sequences of points captured between a stylus' pen-up and pen-down events. Roughly speaking, sketched symbols are sets of line primitives organized spatially and sometimes in a temporal sequence. These characteristics make desirable to use structural approaches to recognize drawings. In this paper we focus on a syntactic approach to describe and recognize graphical symbols in an on-line framework. A syntactic approach addresses two relevant issues: the description of the graphical entity and the recognition process. The former is based on the theory of formal languages, where a grammar

W. Liu, J. Lladós, and J.-M. Ogier (Eds.): GREC 2007, LNCS 5046, pp. 169–180, 2008.

describes the recognized shapes and its productions represent the relations among composition elements. The latter requires a parsing approach. A parser is a process that, given an input and a grammar G, says if the input belongs to the language generated by the grammar, L(G).

A number of grammatical formalisms exist in the literature to describe bi-dimensional graphical objects. Early attempts augmented linear languages with 2D operators to express the spatial relations among the primitives. Picture Description Languages or Plex grammars [7] are two examples of this approach. Over the last two decades new paradigms of languages have been studied. These languages are inherently bi-dimensional, and thus more apt to describe 2D symbols. They are referred as Visual Languages (VLs). Among the different approaches to VLs we find Relational Grammars [6], where productions describe relations among the different primitive symbols in terms of a set of attributes defined as join points. Adjacency Grammars [2] define a set of constraints denoting spatial, temporal or logic relations. Graph Grammars [19] define productions in terms of graph-based rewriting rules but require complex rules.

Together with grammatical formalisms parsing paradigms were devised to validate whether visual languages belonged to the language generated by those grammars. Most of these parsing methodologies are tailored to a specific grammatical formalism, such as the parsers presented in [16], [18]. Despite these specific methodologies other works extend traditional parsing techniques to try and develop more general methods such as the work of Costagliola et al. [21] which extends conventional LR-parsing techniques to the realm of visual languages.

As described above, a syntactic approach to sketched symbol recognition requires first a grammatical model and second a parsing engine to perform the proper recognition. To this end, we adopt an Adjacency Grammar to describe 2D shapes using a linear language by defining constraints to describe the different relations among the parts composing a sketch. Then we use an incremental parser, to analyze visual sketched sentences. This is done by constructing a parsing-tree each time a new token is drawn. Differently from traditional parsers, our parsing algorithm is able to cope with the main issue of VLs, that is, parsing the input in an order free manner. In this way, the relations the parse tree is built according to spatial or logical relations among the different symbols composing a sketch rather than relying on their temporal sequence as happens with conventional textual languages. This is because, our parser uses a spatial data structure to allocate the different tokens as they are analyzed and, to search the set of neighbouring symbols to match grammatical rules when a new token is drawn and recognized.

This paper is organized as follows: section 2 presents related work on sketch recognition systems and syntactic approaches to describe 2D patterns. In section 3 we present the syntactic approach used to describe and interpret sketches. Section 4 presents experimental evaluation of our work. Finally, section 5 discusses these results and points to future directions in our research.

2 Related Work

This section discusses related work developed in the field of sketch recognition and compares syntactic approaches to describing 2D symbols.

2.1 Sketch Recognition

Sketch recognition is a field of increasing interest due to the progress of digital pen devices allowing interaction between Humans and Computers. This requires designing new applications to analyze sketched inputs, either statically, as in document analysis and recognition or more recently in interactive settings, including calligraphic and pen-based interfaces. In the literature, we find different work that describes sketch recognition as applied to different fields. Landay and Myers presented SILK [13] a system to describe User Interfaces. SILK system attempts to recognize basic primitives forming the sketch using Rubine's algorithm [14] to recognize gestures. Once a primitive is detected the system tries to detect the spatial relations between the primitive and other components or widgets. The system then returns the recognized widget to the user, who is able to correct this output if an error occurs. However, using Rubine's recognizer limited the system to single-stroke basic primitives.

Hammond and Davis [15] developed a system to recognize UML Class Diagrams in four steps: pre-processing, selection, recognition and identification. The pre-processing step classifies the most recent stroke in one of four categories an ellipse, a line, a polyline or a complex shape. Next, the system attempts to match this to a set of (previously drawn) unrecognized strokes. The authors limit the match candidates to at most nine trying to avoid an exponential time search on the number of strokes required to identify a symbol. This task is ascribed to specific symbol recognizers which identify each different symbol allowed in UML diagrams. The identification process combines the probability of each recognizer with other criteria, such as the number of strokes that compose the symbol recognized.

Kara and Stahovic [11] developed a sketch recognition system to describe engineering circuits that can be used as input to Simulink, using a hierarchical recognition approach. First, the system tries to identify specific symbols as markers. These symbols should be easy to recognize and serve as anchors to help recognizing the remainder of the sketch. In this case, the markers describe arrows which are recognized according to features based on drawing speed. Then the system generates a set of symbol candidates, by taking into account the number of input and output arrows for a given cluster. To recognize a symbol among the different candidates the authors use the combination of four different recognizers by choosing the answer with the best score.

Alvarado and Davis in [10] present a multi-domain sketch recognition system, based on a dynamically constructed bayesian network. The network is constructed using LADDER [8] a language that is able to describe and draw shapes for a specific domain. For each new stroke is drawn the system classifies it in one of the basic categories. Then a hypothesis is generated in three steps: a bottom-up step generates the hypothesis from the new stroke, followed by a top-down step that attempts to find subshapes missing on the partial hypothesis created by the previous step and finally, a pruning step that keeps the number of hypothesis manageable to be analyzed in real time.

2.2 Grammatical Symbol Recognition

Different grammatical formalisms have been proposed to describe visual constructs (symbols). These formalisms describe symbols as a collection of basic primitives and a set of relations connecting those shapes.

Linear grammatical formalisms are presented in [16], [17] and [18]. These three grammars describe productions as a set of symbols and a set of constraints among those symbols. While *Relation Grammars* [17] define their productions over a set of un-attributed tokens, *Constraint Multiset Grammars* [16] and *Picture Layout Grammars* [18] describe their productions using sets of attributed tokens. *Constraint Multiset Grammars* are context-sensitive grammars that define a set of tokens that may exist to produce a valid rule. In this latter formalism, contextual tokens may be specified in spatial or logical constraints, while not being considered part of the production proper.

Coüasnon developed a language named EPF (Enhanced Position Formalism) [20] using an operator based grammatical formalism. In EPF the productions are concatenations of symbols and operators between the symbols. The operators may describe positional relations, factorization of symbols, etc.

While the grammatical formalisms presented before are linear grammars, other researchers focus on visual languages defined via high-dimensional grammars. Wittenburg and Weitzmann [6] present Relational Grammars. These grammars are high-dimensional context-free grammars. Grammar productions are defined over ordered sets of symbols and a set of constraints among the symbols. Other high-dimensional formalisms include Layered Graph Grammars [19]. These grammars are context-sensitive, restricting the size on the left-hand of the production to be smaller than the right-hand.

More recent work that combines sketch recognition with a syntactic approach. Sketch Grammars [21] extend context-free string grammars, by defining relations other than concatenation relations. These relations include temporal or spatial constraints among symbols. Productions in these grammars also define a set of actions that may include drawing constraints and semantic context. LADDER [8] is a closely related technique that describes relations among complex symbols defined in terms of basic elements and specifies drawing, editing and semantic actions as part of a production.

3 A Syntactic Approach to Recognize Hand-Drawn Sketches

The syntactic formalism presented in this paper is a one-dimensional grammar based on Adjacency Grammars [2]. The symbols in the right-hand-side of a production are described as an unordered set of tokens that should obey a set of constraints, which allows these grammars to describe drawings in an order-free manner.

Adjacency Grammars are formally defined as a 5-tuple G = $\{V_t$, V_n, S, P, C$\}$ where:

- V_t represents the terminal vocabulary.
- V_n represents the non-terminal vocabulary.
  With $V_t \bigcap V_n = \emptyset$ and $V_t \bigcup V_n = \Sigma$ being Σ the alphabet of the language generated by the grammar L(G).
- S is the start symbol.
- P is the set of productions of the grammar defined as:

  $\alpha \rightarrow \{\beta_1, \ldots, \beta_j\}$ if $\Gamma_1(\Phi_1, c_1), \ldots, \Gamma_n(\Phi_n, c_n)$

  Where $\alpha \in V_n$ and $\forall i \in [1, \ldots, j] \beta_i \in \{V_t \cup V_n\}$, constitute the possible empty multiset of terminal and non-terminal symbols. $\forall k \in [1, \ldots, n]$ Γ_k are the

adjacency constraints defined on the attributes of the subsets $\Phi_k \subset \{\beta_1, \ldots, \beta_j\}$ and c_k are the cost functions associated to each constraint.

– C represents a set of constraints. This set represents the different spatial relations that we may find between two different grammatical symbols.

f_α: $R_1^d \times \ldots \times R_j^d \rightarrow R_\alpha^d$ is a function that calculates the attributes of the new token from the attributes of the tokens of the right hand of the grammatical productions. Being *d* the cardinality of the attributes of the token α and *j* the number of tokens on the right-hand-side. This function is used when a symbol is reduced from a grammatical production during the parsing process.

Concerning the parser required in this approach, it is an incremental on-line parser to recognize sketches, which analyzes each new input token drawn until all inputs are processed. Then the parser either recognizes the whole sketch, or signals an error due to a invalid input.

While conventional parsers analyze input tokens with these grammars according to a predefined input order, in a sketching framework we can not expect such an ordered list of tokens when a user is drawing a symbol. On the contrary, each user may draw the constituents of a symbol in a different order. There are two solutions to this problem. The first establishes a predefined input order, and the user has no freedom to draw a sketch which allows a conventional linear parsing to be applied. The second works with no predefined order but entails a high computational cost because, for each new token, the parser has to look among all the previously drawn symbols for those that may be combined to yield a valid rule.

The parser presented in this paper requires no predefined order in the input but uses a uniform grid to avoid re-scanning all the input as each new token is drawn. When a new token is read by the parser, it is placed into an array of rectangular cells. Then the parser searches the neighbouring cells for symbols that may produce a valid rule, instead of searching all symbols seen so far. For well-behaved languages this allows polynomial-time search while making possible to provide an analysis in real time for reasonable input sentences.

Constructing the grid entails some decisions: whether it should be a *static* or *dynamic* structure, the size of the cells, whether cells should be of fixed size or use adaptive dimensions, and how to place the symbols into the grid. Using a *dynamic* grid requires recalculating the regions each time that a new item is inserted. Regarding the size of cells, if we choose too small a size, each inserted symbol will be stored in many small cells, thus taking up more space, and the parser will need to analyze more cells to find candidate tokens, taking up more time. Using larger cells means that a major number of symbols could be stored inside any given cell. This may mean a large number of ”false candidates” showing up in neighbor queries, thus wasting computational resources. Finally, we need to take into account the method used to calculate the cells each symbol belongs to. Using a bounding-box is not always an adequate heuristic to find the cells spanned by a given symbol. I.e. if we have a token that describes a diagonal line its bounding box will intersect many cells that do not really belong the symbol. This will then waste computational resources, by attempting to match the symbol against non-neighbor terms.

(a) (b)

Fig. 1. Different placements algorithms used on the grid based parser:(a) Bounding-Box and (b) Bresenham's Algorithm

Being:
$Gr = \{Cx_1, \ldots, Cx_{n \times m}\}$ the Grid over the space, with each of its Cx_i cells.
$BB(Cx_i)$: The boundind box of the cell Cx_i $(x_1, y_1), (x_2, y_2)$
$N(Cx_i)$: The set of Neighbouring cells of Cx_i
$E(Cx_i)$: The elements (tokens, completed or unfinished rules) inside a cell Cx_i
$E(Cx_{i_1}, \ldots, Cx_{i_m})$: The elements (tokens, completed or unfinished rules) inside a set of i_m cells
t is a token
$Cells(t)$ is the set of cells that belongs to the token t following fig. 1.
$r' = Validate(r, e_1, e_2)$ where r' is the modification of the rule r if $e_1 and e_2$ satisfy the constraints defined in r.
$r' = Finalize(r, e_1)$ add the element e_1 if it satisfies the constraints defined in r.
$Valid(e)$ returns true if the element e has all of its composing elements and they have produced a valid reduction. Given a token t

```
Parser(t)
  C = Cells(t)
  N = {Cx_i, ∀Cx_j ∈ C&&Cx_i ∈ N(Cx_j)}
      For each t' ∈ E(N)
          If Valid(t') then
              Find productions Pr with t and t'
              For each p' ∈ Pr
                  t'' = Validate(p', t, t')
                  insert t'' into Cells(t'')
                  If Valid(t'') then
                      Parser(t'')
                  End If
              End For
          Else t'' = Finalize(t', t)
          If Valid(t'') then
              Parser(t'')
          End If
      End For
```

Fig. 2. Incremental Grid-Based On-line Algorithm

In our method after some experiments we have defined a static grid with a fixed cell size of $1/2 \times 1/2$ inch. Using larger cells involves covering a more extensive area and therefore taking more primitives into account. On the contrary, smaller cells entail a larger number of memory accesses. To place the symbols into the cells we decide to use two different methods depending on whether the token is a line or not. If the token is a line we use Bresenham's algorithm [9] to compute the cells that contain the line. On the contrary, the bounding box determines the set of cells belonging to it, see fig. 1.

Our parser works as described in fig. 2: When a new primitive p, is input, the parser finds out which grid cells it overlaps. Then the parser searches among the neighbouring cells which contains graphical elements that, together with the new primitive, can match a production in the grammar. If no such elements are found and p does not match the right-hand-side of a production by itself, the parser looks for the next primitive input. If a matching production is found, the parser will check to see if the constraints on the right-hand-side are met. If all the constraints match, the symbol s_1 on the left-hand-side of this rule, generates a parse item, which is placed into the grid. Otherwise, if part of the production rule is valid but there are still missing components s_1 is marked as an

unfinished parse item and it is placed also into the grid. Note that incomplete parse items can match incoming primitives as a production would do. If a complete parse item has been produced, we check to see if additional productions can match s_1 (the non-terminal labelling the new item). We recurse on new (completed) items until no more production rules can fire.

4 Experimental Evaluation and Discussion

To evaluate our syntactic approach, we have defined two example grammars, as shown in fig. 3. The first grammar, in fig. 3.a, describes architectural floor-plans. Here, a room is defined as a *rectangle* which has a *door* and a *window* that intersects with it. The second grammar describes a graphical user interface GUI as shown in fig. 3.b. For example, we define a MenuBar as a *rectangle* that contains two or more Menu tokens. The basic primitives of these grammars are described by Adjacency constraints. To simplify writing these grammars, the definition of these basic primitives is automatically done using a grammatical inference method (see [5]).

The experiments show how our grammatical formalism is able to describe hand-drawn sketches using two grammars described above. The first experiment highlights the resource savings due to the adoption of a grid and which is the correct size of its cells. The second one shows how our method is able to cope with distortion, errors and uneven spatial distribution of tokens. Finally, we have tested the syntactic approach with sketches drawn by different users.

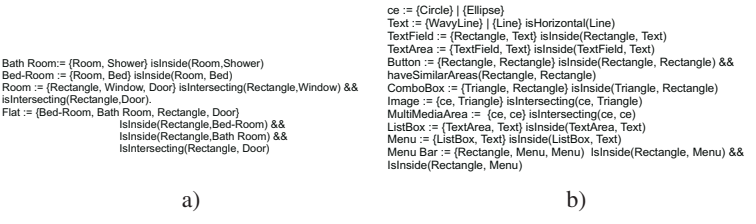
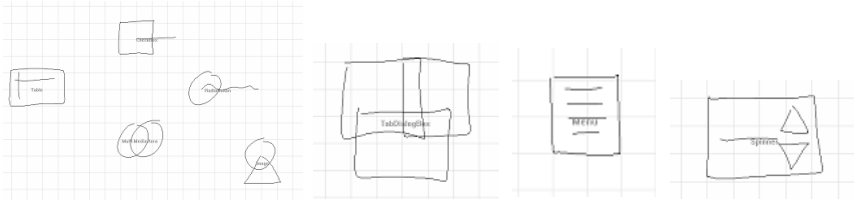


Fig. 3. Example of a grammar specification

4.1 Experiment 1: Grid Tuning

This experiment evaluates the computational resource savings afforded by the grid and allows us to estimate a good cell size. Table 1.a shows number of memory accesses by our parsing algorithm. As we can see a small cell size increases the number of memory accesses. Such grids involves small size on the cells that may introduce errors at the time to search symbols in the neighbouring cells of the new token, as two real neighbouring tokens may be placed in not neighbouring cells. This requires expanding the search area to encompass more cells, which in turn increases again the number of memory accesses. On the other hand, large cells yield more ”false positive” tests, as more elements which



a) b) c) d)

Fig. 4. Examples used on grid tuning: a) Example1, b) Example2, c) Example3, and d) Example4

are placed too far apart to match adjacency relations show up in neighbor queries. As can bee seen from Table 1, the smallest number of memory accesses seem to occur around 1/2 in x 1/2 in cells. This, of course is dependent on input data average size and primitive types. Different input tokens would probably require cells of a different size. Further more comprehensive testing would be required to establish better heuristics for cell size which would take into account grammar tokens and input characteristics.

Table 1.b shows a comparison in terms of finished and unfinished parse items between our method and a non-constrained (i.e. griddles) parser. This parser is somewhat similar to the parser defined by Golin in [18]. We consider as unfinished parse items those productions for which the parser has not been able to match all symbols on the right-hand-side. As we can see, the difference between the two methods lies in the number of unfinished productions. When the user draws a new stroke, unconstrained parsers have to re-analyze all the primitives previously seen, even if they bear no relation to the symbol being drawn. Our method takes into account less primitives, thus it reduces the number of feasible productions tested while generating less temporary (unfinished) parse items. Finally, table 1.c shows the recognition times for each of the samples. As we can see, the time values are reasonable to analyze an input in real time.

Table 1. Parser performance a) Number of memory accesses as a function of cell size, b) Number of Finished and Unfinished parse items created and c) Recognition Time

Sample \ Cell Size	1 inch	1/2 inch	1 cm	1/2 cm
$Example1$	360	504	774	2529
$Example2$	189	333	504	1944
$Example3$	108	108	180	450
$Example4$	126	144	378	945

a)

Sample	With Grid Finish./Unfinish.	Without Grid Finish./Unfinish.
$Example1$	$15/1$	$17/125$
$Example2$	$1/2$	$1/65$
$Example3$	$8/4$	$8/86$
$Example4$	$2/1$	$2/67$

b)

Sample	#tokens	#symbols	Time(ms)	Average(ms)
$Example1$	19	15	404	26.93
$Example2$	12	1	155	155
$Example3$	8	8	220	27.5
$Example4$	11	3	203	67.3

c)

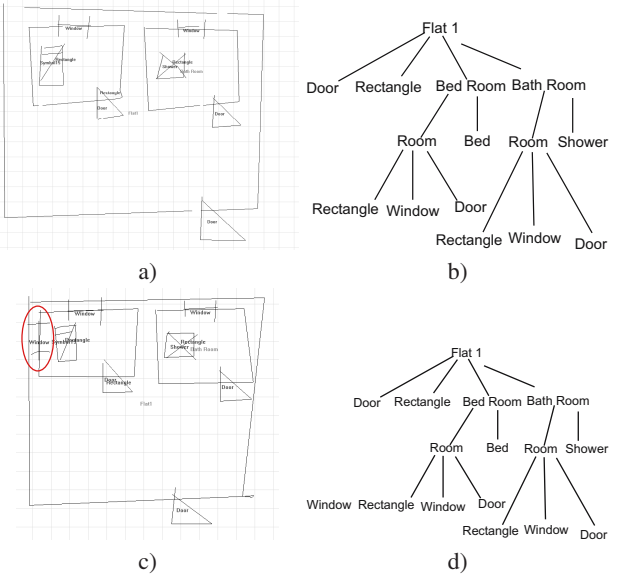


Fig. 5. Architectural floor-plan a) and corresponding parse tree b) c)Another Instance and its parse tree d)

4.2 Experiment 2: Distortion Tolerance

Contrary to document recognition, error tolerance and the ability to cope with sizeable variations in input tokens are very desirable features to have in a parsing algorithm tailored to interactive use. This experiment shows how tolerant our method is to distortion and input errors such as added (extraneous) elements and changes in spatial relationships among the strokes on a given sketch.

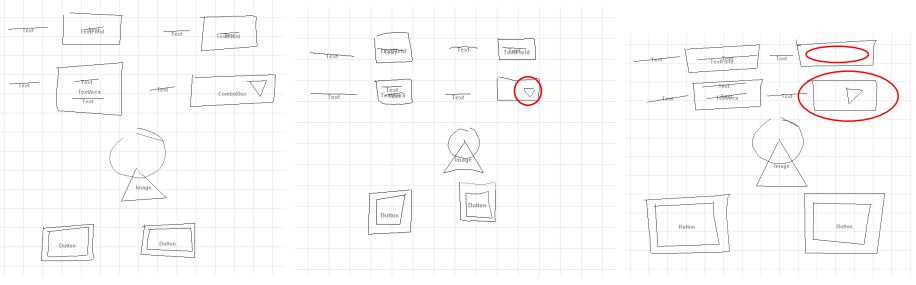
Figure 5 shows two samples representing sketched floor-plans and their corresponding parse trees. As we can see, at the time that each new token of the input of the left-hand is drawn by the user, the parsing algorithm constructs the parse tree on the right side where the leaves corresponds to the terminal symbols of the alphabet and each non-leaf node represents a non-terminal symbol labelling the corresponding production in the grammar. Figure 5.c shows the same visual sentence where an additional element circled in red has been added. This element has been recognized as a window (as described by the grammar in Fig. 3.a), it does not satisfy the relational constraint to the other elements of the production. Although this additional symbol appears to be spurious input, we can see that the parser is able to describe the whole sketch, and with some extra work, it would be possible to identify which strokes do not participate on a complete parse tree (i.e. one labelled by the start symbol at the root). This is accomplished through cover sets as described in [1] .

4.3 Experiment 3: User Testing

This experiment evaluates the ability of our methodology to accommodate the variability in drawing styles produced by different users. To perform the experiment we

Table 2. Experimental test based on a set of users

Sample	# Recognized tokens	# Unrecognized tokens	Error Description
1	13	1	Constraints Failure on the token window.
2	15	0	No problems into the sketch.
3	15	0	No problems into the sketch.
4	15	0	No problems into the sketch.
5	12	3	Window recognized as a Rectangle.
6	15	0	The sketch does not present problems
7	15	0	The sketch does not present problems
8	15	0	The sketch does not present problems
9	15	0	The Sketch does not present problems
10	19	0	The sketch presents some extra tokens forming a bed-room that has not been recognized.



a) b) c)

Fig. 6. Three different samples from Sketched User Interfaces a) Complete Recognition, b) token mis-recognized and c) missing token

showed a sketched floor-plan, see fig. 5.a, to ten users who were then asked to draw a similar sketch. Each floor plan requires drawing roughly 40 strokes to depict 15 visual elements. We ran our parsing algorithm on these sketches. Out of the ten sketches, two were not correctly interpreted. Seven sketches contain the forty recognized strokes with a correct interpretation. On one case the user repeated a token, due to a mis-recognized production (constraint failure). Table 2 summarizes the experimental outcome. From the results obtained by the parsing methodology we can see that our approach is able to interpret floor plans interactively. Indeed, some of the failures can be corrected by adding interactive correction capabilities to our method. This will be done in future versions of the system.

We also tested the syntactic approach presented in this paper with a grammar that describes User Interfaces. Figure 6 shows three samples of the same sketch drawn by three different users. Figure 6.a shows a sketch where all the tokens are well recognized comparing it with the sketches in fig. 6.b and fig. 6.c we can see that in the first only one token is not recognized. In this case the unrecognized token is due to a stroke preprocessing error. The system has recognized an arc instead of two segments. The

sketch in fig. 6.c has one token missing on its top part and there is a token that has not been well recognized. This is due to the fact the token was highly distorted.

5 Conclusions

In this paper we have presented a syntactic approach for sketch recognition including two components: First, we have described a grammatical formalism based on an Adjacency grammar that allows describing spatial relations among the different symbols that compose a hand-drawn sketch. Second, a parsing method based on a regular grid has been introduced. This parsing method allows partially re-scanning the input when new strokes are entered by users, which makes it suitable for interactive use. Preliminary experimental evaluation of the system shows that using a uniform grid with cells of adequate size reduces the spatial and temporal complexity of the parsing algorithm, thus making it suitable to be used in an interactive sketch recognition framework. Moreover, the syntactic approach presented is flexible enough to describe and interpret sketches from different problem domains, in this case architectural floor-plans and User Interfaces. Finally, the results obtained from the evaluation suggest a good scalability to larger drawings. As future work we plan to expand the system to make it more flexible and support interactive user correction and modification of input drawings.

Acknowledgements

This work has been partially supported by the Spanish project TIN2006-15694-C02-02, the consolider project CONSOLIDER-INGENIO 2010 (CSD2007-00018) and by Portuguese Science Foundation grant POSC / EIA / 59938 / 2004 (DECORAR).

References

1. Jorge, J.A.P.: Parsing Adjacency Grammars For Calligraphic Interfaces. Rensselaer Polytechnic Institute, New York (1995)
2. Jorge, J.A.P., Glinert, E.P.: Online Parsing of Visual Languages Using Adjacency Grammars. In: 11th International IEEE Symposium on Visual Languages, pp. 250–257 (1995)
3. Lladós, J., Valveny, E., Sánchez, G., Martí, E.: Symbol Recognition: Current Advances and Perspectives. In: Blostein, D., Kwon, Y.B. (eds.) GREC 2001. LNCS, vol. 2390, pp. 104–127. Springer, Heidelberg (2002)
4. Wenyin, L.: On-line Graphics Recognition: State-of-the-Art. In: 5th IAPR Workshop on Graphics Recognition, Barcelona, pp. 291–304 (2003)
5. Mas Romeu, J., Lamiroy, B., Sanchez, G., Llados, J.: Automatic Adjacency Grammar Generator from User Drawn Sketches. In: 18th International Conference on Pattern Recognition, Hong-Kong, pp. 1026–1029 (2006)
6. Wittenburg, K., Weitzman, L.: Relational Grammars: Theory and Practice in a Visual Language Interface for Process Modelling. In: International Workshop on Theory of Visual Languages, Italy (1996)
7. Bunke, H.: Hybrid Pattern Recognition Methods. In: Bunke, H., Sanfeliu, A. (eds.) Syntactic and Structural Pattern Recognition.Theory and Applications, pp. 307–347. World Scientific Publishing Company, Singapore (1990)

8. Hammond, T., Davis, R.: LADDER: A Language to Describe Drawing, Display and Editing in Sketch Recognition. In: Internaltional Joint Conference on Artificial Intelligence, Acapulco, pp. 461–467 (2003)
9. Bresenham, J.: Algorithm for Computer Control of a Digital Plotter. IBM System Journal 4(1), 25–30 (1965)
10. Alvarado, C., Davis, R.: Dinamically Constructed Bayes Nets for Multi-Domain Sketch Undrestanding. In: Internaltional Joint Conference on Artificial Intelligence, San Francisco, pp. 1407–1412 (2004)
11. Kara, L.B., Stahovich, T.F.: Hierarchical Parsing and recognition of hand-sketched diagrams. In: 17th Annual ACM Symposium on User Interface Software and Technology, pp. 13–22 (2004)
12. Costagliola, G., Deufemia, V., Risi, M.: A Multi-layer Parsing Strategy for On-line Recognition of Hand-Drawn Diagrams. In: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 103–110 (2006)
13. Landay, J.A., Myers, B.A.: Sketching Interfaces: Toward More Human Interface Design. IEEE Computer 34(3), 56–64 (2001)
14. Rubine, D.: Specifying Gestures by Example. In: 18th Annual Conference on Computer Graphics and Interactive Techniques, pp. 329–337 (2001)
15. Hammond, T., Davis, R.: Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. In: AAAI Spring Symposium on Sketch Understanding, pp. 59–68. Palo Alto, Menlo Park (2002)
16. Marriot, K.: Constraint Multiset Grammars. In: IEEE Symposium on Visual Languages, St. Louis, pp. 118–125 (1994)
17. Crimi, C., Guercio, A., Nota, G., Pacini, G., Tortora, G., Tucci, M.: Relation grammars and their application to multi-dimensional languages. Journal of Visual Languages and Computing 2(4), 333–346 (1991)
18. Golin, E.J.: Parsing Visual Languages with Picture Layout Grammars. Journal of Visual Languages and Computing 2(4), 371–394 (1991)
19. Rekers, J., Schurr, A.: Defining and Parsing Visual Languages with Layered Graph Grammars. Journal of Visual Languages and Computing 8(1), 27–55 (1997)
20. Coüasnon, B.: DMOS, a generic document recognition method: application to table structure analysis in a general and in a specific way. International Journal on Document Analysis and Recognition 8(2-3), 111–122 (2006)
21. Costagliola, G., Deufemia, V., Risi, M.: Sketch Grammars: A Formalism for Describing and Recognizing Diagrammatic Sketch Languages. In: International Conference on Document Analysis and Recognition, Hong-Kong, pp. 1226–1230 (2005)