Multimodal Technologies and Interaction

MDPI

Article

Multi-Domain Recognition of Hand-Drawn Diagrams Using Hierarchical Parsing

Vincenzo Deufemia * and Michele Risi

Dipartimento di Informatica, Università di Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano, Salerno, Italy; mrisi@unisa.it

* Correspondence: deufemia@unisa.it; Tel.: +39-089-96-3346

Received: 30 June 2020; Accepted: 12 August 2020; Published: 14 August 2020

Abstract: This paper presents an approach for the recognition of multi-domain hand-drawn diagrams, which exploits Sketch Grammars (SkGs) to model the symbols' shape and the abstract syntax of diagrammatic notations. The recognition systems automatically generated from SkGs process the input sketches according to the following phases: the user' strokes are first segmented and interpreted as primitive shapes, then by exploiting the domain context they are clustered into symbols of the domain and, finally, an interpretation of whole diagram is given. The main contribution of this paper is an efficient model of parsing suitable for both interactive and non-interactive sketch-based interfaces, configurable to different domains, and able to exploit contextual information for improving recognition accuracy and solving interpretation ambiguities. The proposed approach was evaluated in the domain of UML class diagrams obtaining good results in terms of recognition accuracy and usability.

Keywords: diagram understanding; freehand sketching; sketch recognition; multimodal interfaces; visual language parsing; grammars

1. Introduction

Hand-drawn sketches allow users to quickly draw new technical solutions and effectively communicate them to a variety of audiences. The diffusion of sketch-based hardware devices has increased the interest in the creation of computer software that works exclusively on freehand drawings [1–5], enhancing the efficiency and effectiveness of user interfaces [6,7]. Sketch recognition can also capture the sketching component of a multimodal conversation about design. However, the existing sketching systems have limited recognition capability for the input drawings, which makes difficult the conversion of a sketch directly into formal, domain specific models for more powerful design systems. This introduces redundancy and inefficiency when the systems are used as early phase design tools, making sketching not a homogeneous part of the design process.

The sketchy graphic objects drawn using a digital pen are not usually easy for machines to understand and process. Sketch recognition algorithms can simply clean up roughly drawn strokes [8], in this case they are called sketch beautification algorithms [9,10], and/or solve the problem of recognizing the symbols in the hand drawn user's diagrams. This problem is particularly difficult, since the symbols of a sketched diagram can be drawn by using a different stroke-order, -number, and -direction. Several work concentrates on the recognition of symbols formed by a single-stroke gesture [11], also named glyph, and on the recognition of gestures formed by multiple strokes [12]. The recognition of continuous sketches also involves the activities of segmentation and clustering of the user's strokes at the same time, where the segmentation has the goal of decomposing an input sketch into several semantically meaningful components [13,14]. Because this problem may require a considerable amount of time to be solved, the systems implementing such a type of recognition

work under some assumptions about how the sketches are drawn [15–17]. The difficulties in the recognition process are also increased by the lack of precision and the presence of ambiguities in messy hand-drawn sketches. Indeed, hand-sketched symbols are imprecise in nature such that corners are not always sharp, lines are not perfectly straight, and curves are not necessarily smooth.

This paper presents a sketch recognition system for multi-domain hand-drawn diagrams based on grammar formalisms and parsing technologies [18,19]. In particular, the approach exploits Sketch Grammars to hierarchically define both the symbols' shapes and the abstract syntax of diagrammatic notations, and to automatically generate efficient LR-based parsers [20]. The latter are on the top layer of the sketch recognition strategy, while the bottom layer consists of a domain-independent primitive shape recognizer. Once integrated into a sketch editor, the proposed sketch recognition system performs a fine-to-coarse, incremental, and unobtrusive interpretation of the sketches as they are created. In particular, at a fine-grain level, the input strokes are decomposed and analyzed as primitive shapes, and then grouped into domain symbols according to grammar-based symbol recognizers; at a coarse-grain level, the best interpretation of the input drawings is determined using the contextual information provided by the recognizer of the diagrammatic notation.

The main contribution of this paper is an efficient model of parsing that (i) can be used both in offline sketch recognition based on document image analysis and in on-line or interactive sketch recognition based on the interpretation of digital ink strokes; (ii) can be configured to different domains; and, (iii) is able to exploit contextual information for improving recognition accuracy and solving interpretation ambiguities.

The proposed approach integrates a number of desirable features: generality, robustness, and efficiency. In fact, Sketch Grammars allow users to construct recognizers of any diagrammatic or iconic domain. These recognizers do not constrain users to a specific drawing style, since the symbols of the diagrams can be drawn with a varying number of strokes and without a particular direction or order. The diagrams can also be draw in interspersed fashion, i.e., new drawn symbols are recognized, even though other symbols have not been completely drawn. The robustness of the approach resides in the continuous interaction between the two top layers allowing for an extended recognition process. In particular, a layer performs the recognition of symbols in isolation generating a number of possible interpretations, while the top layer uses the domain knowledge to resolve inherent interpretation ambiguities and/or to correct misrecognized or incomplete symbols. This is possible thanks to the context in which the symbols appear, which influences the recognition of the other symbols. Moreover, the proposed parsing technique includes error recovery techniques, as developed for programming language compilers [21], to deal with the inaccuracy of hand-drawn symbols.

The efficiency of the proposed approach was tested in a variety of application domains. In this paper, we present the results obtained on the domain of UML class diagrams. We also compare the achieved results with those that were obtained by a non-hierarchical parser whose disambiguation process is based on partial contextual information of the symbols. Moreover, we have evaluated in the same domain the usability of the tool implementing the proposed recognition model, namely *SketchBench*, obtaining better results in terms of easiness of use and pleasantness to use than traditional WIMP (Windows, Icons, Menus, and Pointing) CASE tools.

The outline of the paper is as follows: Section 2 describes the challenges of recognizing real world sketches. Section 3 discusses previous work. Section 4 illustrates the main challenges in the construction of diagrammatic sketch recognizers and introduces the proposed approach. Section 5 describes the Sketch grammar formalism, whereas the recognition strategy is introduced in Section 6. The evaluation of the recognition system and the usability study of *SketchBench* are presented in Section 7. Finally, the conclusion and further research are outlined in Section 8.

2. Sketch Recognition Challenges

As opposed to traditional diagram editors, where the domain specific symbols are provided in palettes, abstract shapes are imprecise and highly variable in hand-sketched diagrams. Thus, the first

challenge in building a sketch-based system is recognizing the meaningful patterns that are 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 and sensitive to the fact that too much variation might change the identity of the pattern. As an example, let us consider stroke s_1 in Figure 1. By itself it is an arc, but in the recognition process it should also be considered as a line segment, since these two primitive elements look very similar when drawn. Moreover, sketching, as well as all other natural ways of conveying information, is an intrinsically ambiguous 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. Furthermore, semantically different objects might be graphically represented by identical or apparently similar symbols. As an example, a square symbol in a class diagram might be both the border of a class symbol and the diamond of the aggregation relationship, as shown by strokes s_3 and s_2 in Figure 1.

Sketch recognition systems should also not place constraints on how the users can draw a symbol. Indeed, they should be able to draw without having to worry about where to start a stroke, how many strokes to use, in what order to draw the strokes, and so on. Thus, the recognition should be independent of stroke-order, stroke-number, and stroke-direction, as well as invariant to rotation, scaling, and reflection of symbols. The independence of the sketch recognition process from the number of strokes is achieved by ink parsing whose goal is to establish which strokes are part of which shapes by grouping and segmenting the user's strokes into clusters of intended symbols [22]. This is a particularly challenging problem, since the same shape can be drawn by varying the number of pen strokes. For example, the border of a class symbol can be drawn as a single pen stroke (see stroke s_3 in Figure 1), or as two or more separate strokes (such as strokes s_4 and s_5 in Figure 1). Alternatively, a single pen stroke can contain multiple shapes, such as stroke s_6 in Figure 1 representing an association symbol (the horizontal arrow) and the border of a class symbol. This variation in drawing style together with the inherent ambiguities increase the difficulties in the segmentation and clustering tasks because a recognizer cannot know a priori how many strokes will be used to draw each object, or the order in which the parts of a shape will appear.

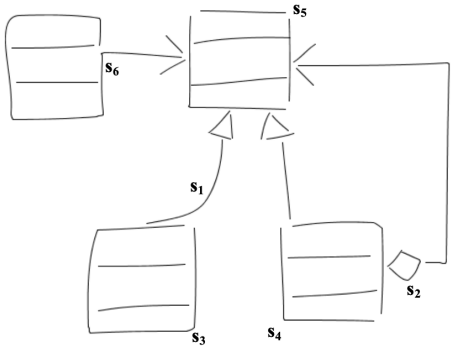


Figure 1. A sketched class diagram.

Another issue is the presence of overtraced strokes in the input sketches. They are created when users draw on top of previous strokes in order to emphasize an aspect of the drawing, tidying up a previous stroke, making a faint stroke darker, and so on [23]. The sketch recognizers must attempt to remove or join these strokes together to recognize the shape correctly. This, in turn, requires the system to determine whether two strokes are intentionally overtraced or simply drawn close together.

In the case of offline sketch recognition, whose process is based on the information extracted from scanned images only, further challenges have to be considered. In particular, the input images have to accurately smoothed and segmented into primitive shapes without take advantage of temporal information.

3. Related Work

In the last decades a considerable amount of work has been devoted to sketch recognition. They can be categorized in different ways, such as based on the information used for the recognition (stroke features, stroke appearance, stroke role), on the modality of recognition (eager vs. lazy) and the kind of feedback, on the imposed restrictions (single stroke vs. multi-stroke vs. multi-symbol), and so on. Three main categories can be identified if we consider the information used for the recognition process: stroke-based, global shape properties, and visual appearance. The recognition that is based on strokes is founded on the assumption that each stroke has a specific role in representing a sketch, thus the recognition methods consider each stroke to determine what role it plays. On the contrary, by using the general properties of shapes and their underlying strokes the assumption that each stroke plays a specific role is relaxed [24–26]. Finally, appearance based methods disregard the individual strokes and focus on the appearance that those strokes represent [27]. These approaches follow the strategy used for image recognition, which is, using hand-crafted features, such as GF-HOG [28] and Fisher Vector [29]. However, they still have a gap in recognition accuracy compared with human performance. Other approaches apply discriminative graphical models to the problem of sketched symbol recognition [30]. Recently, deep convolutional neural networks (DCNNs) have been effectively explored for recognizing sketch objects [31–33].

Other relevant studies, which are complementary to sketch recognition systems, regard the work on perception in sketching interfaces. In particular, the system proposed in [34] determines objects in a sketch grouping the strokes according to the Gestalt principles. In [35], Lank and Saund present a model of users' pen-based selection gestures to inform the design of a faster, more accurate selection mechanism. The approach that is presented in [36] clusters the strokes into sensible groupings aiming to facilitate the editing of sketched content. Because we are mainly interested in sketch recognition for interactive interfaces, in the following we consider stroke-based approaches that are well suited to this kind of interfaces because they enable systems to display sketch interpretations after each stroke or each group of strokes is drawn. Moreover, we discuss the sketch description languages used by multi-domain approaches for a flexible specification of sketch domains.

3.1. Stroke-Based Sketch Recognition Methods

The Rubine recognition engine is a trainable recognizer for single stroke gestures [11], which does not segment the stroke into primitive shapes preventing the construction of a hierarchical multi-stroke recognition system. Moreover, the set of features used by the recognizer makes the systems user- and style-dependent. This means that the strokes must be drawn as in the training phase. The use of linear classification algorithms also requires extensive training to produce an accurate system.

The approach that is proposed in [37] exploits Hidden Markov Models (HMM) to capture the drawing regularities, which allow to segment and recognize the input sketch efficiently. However, the recognizer requires each symbol to be completed before the next one is drawn. Recently, the same authors have extended their HMM-based approach for recognizing sketches also drawn in an interspersed fashion [16]. However, the recognition process does not take into account spatial or geometric constraints beyond those used to encode stroke sequences. Therefore a sequence of strokes that has the right temporal character but the wrong shape can be classified incorrectly. This is also the limitation of the approach proposed in [38]. Here, the authors reduce the sketch recognition problem to a combinatorial optimization problem by heuristically defining an appropriate objective function.

The symbol recognizer that is presented in [22] is capable of learning new definitions from single prototype examples, and allows users to continuously sketch thanks to a multi-level parsing scheme. Contextual information are used to improve accuracy and reduce recognition times. However, the recognition process is driven by easy to recognize symbols, that they assume to exist always in the sketch. Another recognition approach driven by symbol candidates, arrows in the case of diagrams, has been presented in [39].

The parsing approach presented in [40] uses Bayesian networks combining bottom-up and top-down recognition algorithm. In particular, input strokes are used to generate symbol hypotheses within a Bayesian network. Subsequently, the algorithm actively seeks out parts of those hypotheses that are still missing and fixes the interpretation errors by using context. Unfortunately, the high computational cost of the approach makes it unsuitable for real-time recognition. Similarly, the sketch recognizer that is proposed in [41] is built upon the rule-based system Jess, which requires exponential running time and space in the worst case.

The approach that is proposed in this paper is most related to [40] in the sense that hierarchically arranged parsers and symbol reinterpretation based on contextual information are used. However, the recognition model proposed here is considerably different. Alvarado and Davis represent the possible interpretations of the input sketch with a Bayesian network whose nodes are generated with an exhaustive bottom-up approach that employs several heuristics to reduce their number. Moreover, they use a greedy algorithm to select the best interpretation for the sketch. In our approach, we consider an efficient visual language parsing technique for recognizing both symbols and sketch scenes, which linearizes the input information during the recognition. Conversely from Bayesian networks, the parse trees constructed during the recognition process are effectively updatable during the editing process.

3.2. Sketch Description Languages

In recent years several symbol description formalisms and languages have been introduced to enable a flexible specification of sketch domains.

Shape grammars have been the first grammar formalism for describing shape languages [42], but do not consider several stroke information that may be helpful in their recognition. The non-hierarchical language presented in [43] is capable of describing and recognizing stick figures, but it is not suitable to describe large shapes. A similar description language has been proposed by Pasternak in [44] for recognizing CAD-drawings.

The sketch recognition systems described in [45,46] use variants of visual language grammars to provide contextual information in order to face the inherent ambiguity of sketched drawings. However, these early solutions focused on finding a concise way to specify valid visual structures without considering the ambiguity issue.

LADDER is a language that allows designers to specify how shapes are drawn, displayed, and edited in a certain domain [41]. The language consists of a set of predefined shapes, constraints, editing behaviors and a syntax for combining them. New domain objects are created by specifying the shape descriptions. They do not consider the problem of stroke segmentation in case of multi-symbol strokes.

A statistical visual language model for ink parsing has been introduced in [47]. The approach combines a visual language formalism with a statistical model for disambiguation. A new recognizer is created by writing a declarative context-free grammar for the language, generating a model from the grammar, and training the model on drawing examples. The approach is designed to parse the single components of a composite symbol, rather than parsing entire sketches, and it assumes that shapes are drawn in certain preferred orientations.

In [48], an extension of Stochastic Context Free Grammar (SCFG) is used for mathematical expressions analysis. In particular, HMMs are used to recognize mathematical symbols capturing many variability of handwritten mathematical expressions, whereas a SCFG is used to model the relation between the recognized symbols taking into account only stochastic information.

The Context Driven Constraints Multi-set Grammar (CD-CMG) formalism is designed for eager interpretation of hand-drawn documents [49], providing a statistical approach to locally recognize a shape, and a structural approach to model the spatial relations and the global document structure. To deal with the real-time recognition of more complex sketches, CD-CMG was extended by defining a hierarchy between different layers of interpretation, which allows having alternative exploration strategies [50].

The grammar formalism proposed in this paper allows to construct recognizers for a variety of domains by providing structural descriptions of domain symbols and syntactic information of the domain language. The descriptions of the symbol's shapes include tolerances and discriminant values. The first are used to handle the unintentional deviations that are introduced into the shape caused by the imprecision of hand control, whereas the latter are used both for recognizing incomplete symbols and for aiding the disambiguation process of symbols having a similar shape. The effectiveness of the parser strategy is inherited by LR parsing [20].

4. Overview of Our Approach

Based on the issues that were introduced in Section 2, the implementation of a sketch recognition system requires the application of intelligent techniques for their resolution. For instance, symbol recognition from hand-drawn gestures strongly dependents from the structure in which they are recognized, i.e., its surrounding objects. As a consequence, the ambiguities in the sketches can be solved by exploiting the contextual information around the ambiguous parts.

The approach that is proposed in this paper uses contextual information provided by grammars during sketch recognition. In other words, the symbols that are unique to a context are used to solve pending symbol recognition ambiguities. Moreover, the recognition process considers only the stroke' shape as recognition features, ignoring other information, such as drawing speed and size. This allows users to draw the symbols as they would naturally.

A sketch recognition system is defined by specifying a grammar for each symbol of the domain language, and a grammar modeling the abstract syntax of the language, i.e., the possible relations among the symbols (see Figure 2). For instance, the definition of a state transition diagram (STD) sketch recognizer requires a STD grammar and a grammar for each one of the following symbols: state, initial state, final state, initial and final state, and transition. The Parser Generator produces the recognizers for each of these grammar specifications.

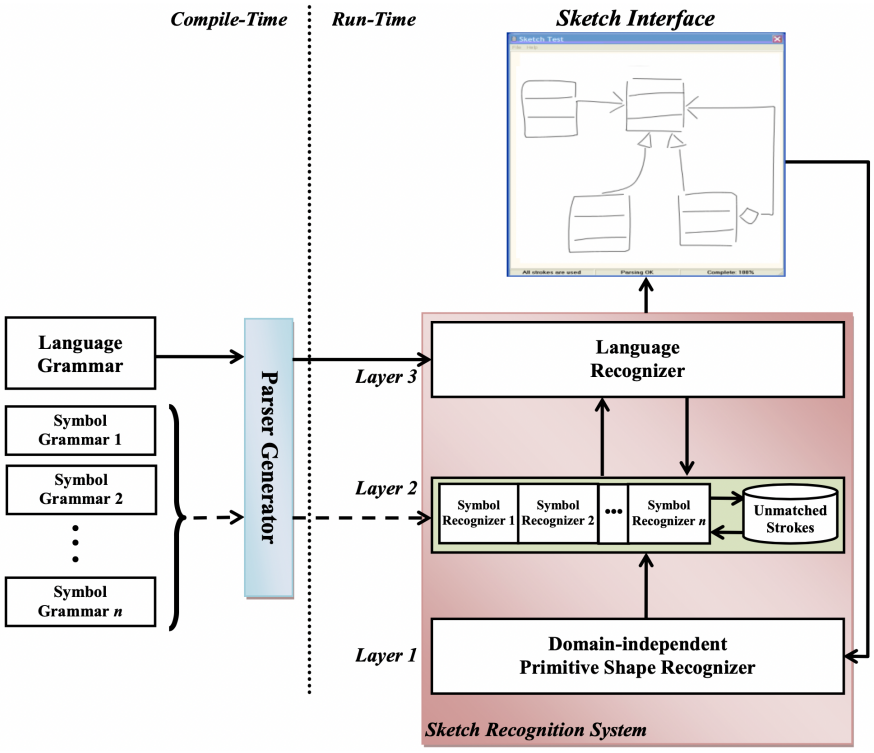


Figure 2. The process of generation of the sketch recognition system.

The input sketches are recognized according to a three-phase process. A domain-independent recognizer at Layer 1 interprets the strokes as primitive objects, such as lines, arcs, ellipses, and so on.

Moreover, the strokes are split by exploiting stroke information such as curvature, speed, and direction, aiming to recognize multi-stroke symbols.

Successively, the symbol recognizers at Layer 2 try to identify possible domain symbols by grouping the primitive objects. In particular, when a symbol recognizer is able to parse a new stroke, it gives as output the new status of the symbol, which can be partially or completely recognized. The unparsed strokes are temporarily stored in unmatched strokes repository of the symbol recognizer, which contains both graphical and classification information.

Finally, the language recognizer at Layer 3 receives the candidate symbols produced by the symbol recognizers, removes some of them based on context, interacts with the symbol recognizers to force the recognition of incomplete symbols, and selects the most suitable interpretation.

The system also works for offline sketch recognition. In this case, the first phase takes an image bitmap as the input sketch. Thus, the domain independent recognizer has to smooth, segment, and convert the image in a set of primitive objects.

This paper considers UML class diagrams as running example [51]. Such diagrams consist of six symbols: class symbol, package symbol, association links, aggregation, composition, and inheritance arrows, which are depicted in Figure 3.

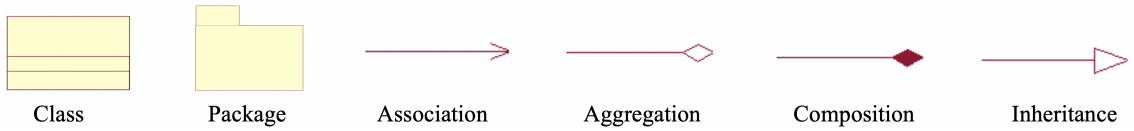


Figure 3. The graphical symbols of UML class diagrams.

5. Describing Diagrammatic Sketches

Sketch Grammars (SkGs, for short) is the formalism used for modeling both the shape of the domain symbols and the abstract syntax of the sketch languages [18].

5.1. Preliminary Definitions

Informally, a sketch language consists of a set of sketch diagrams over a set of domain-specific shapes. The latter can be drawn by varying the number of pen strokes or a single pen stroke can contain multiple shapes. Thus, a sketch sentence of a language L is a set of pen strokes that can be grouped or segmented into shapes that are similar to those in the alphabet, and such that their spatial arrangement complies to the abstract syntax of L. The segmentation process splits pen strokes into primitive shapes, such as line segments and elliptical arcs.

A primitive shape is composed of a graphical representation and a type.The latter has associated a distance function evaluating the similarity of a stroke to the primitive shape, and a set of attributes used to specify relations between primitive shapes. As an example, the type of a line shape may have the least square fitting function as distance function [52], and the position of its end points as attributes.

Let B be a set of primitive shape types with cardinality n. A stroke s is a pair (v, t) where v is a sequence of timed-ordered points $\{p_1, p_2, \ldots, p_M\}$, and t is a sequence of n distance values $f_i(v, b_i)$ with $b_i \in B$, with f_i is the distance function on b_i, and $1 \leq i \leq n$. For brevity, we denote the sequence t with $F(v, B)$.

Two (or more) strokes can be grouped together according to a function cluster defined as cluster $(s_1, s_2) = (v_1 \cup v_2, \mathrm{F}(v_1 \cup v_2, B))$ where $s_1 = (v_1, t_1)$ and $s_2 = (v_2, t_2)$. Whereas, a stroke $s = (v, t)$ can be segmented into $k > 0$ strokes by using an algorithm $seg(s) = (s_1, s_2, \ldots, s_k)$ where $s_i = (v_i, t_i)$, $v = \bigcup_i v_i$ and $t_i = F(v_i, B)$ with $1 \leq i \leq k$ (notice that the definition of stroke may require to be extended in order to include the information required by the considered segmentation algorithm).

The shapes of a domain-specific language can be defined generalizing the definition of primitive shape, i.e., a shape is a pair: a graphical representation and a type. The former is given in terms of primitive shape composition, whereas the latter has associated a set of attributes, which are used

to relate a shape to others, and their values depend on the "position" of the shape in the sentence. For instance, the graphical representation of the class of a UML class diagram is shown in Figure 3, whereas the type is CLASS and the attributes associated to it (attaching points) store the connections to the border of the shape.

A sketch alphabet Σ is defined by a set of shape types. Given the set of primitive shape types B, a sketch sentence on Σ is a sequence of strokes $S_B = \{s_1, s_2, \ldots, s_n\}$, with $s_i = (v_i, F(v_i, B))$ and $1 \leq i \leq n$, such that there exists a clustering and segmentation process that from S_B produces the set of shapes $\{x_1, x_2, \ldots, x_m\}$ whose types are in Σ. For instance, let us consider the strokes depicted in Figure 4b, which represents the class diagram of Figure 1. Figure 4b shows the shapes that are produced by a clustering and segmentation process.

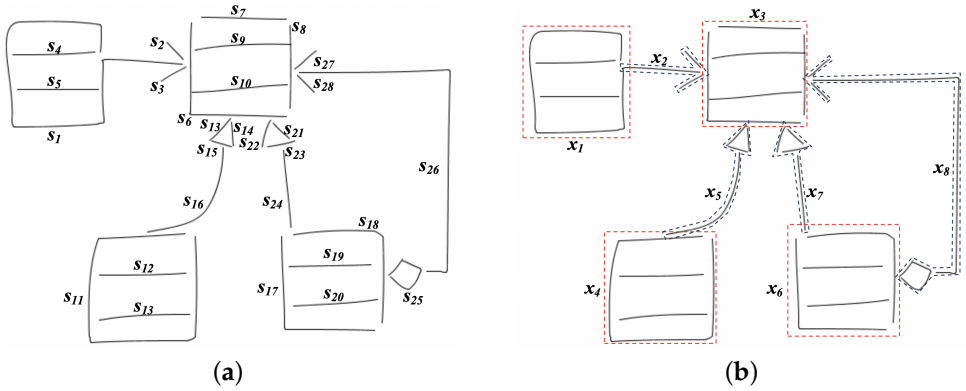


Figure 4. A sketched class diagram (**a**) and the shapes forming it (**b**).

5.2. Sketch Grammars

Grammar productions of sketch grammars define how pen strokes of the input sketches have to be clustered into domain language shapes, and among all possible symbol interpretations to select the most probable ones. The productions embed actions that can be used to visualize the recognized shapes, to specify the editing gesture operations, in order to verify properties on the sketches.

An SkG G can be seen as a particular type of context-free string attributed grammar $(N, T \cup POS, S, P)$, where:

- N is a finite non-empty set of non-terminal shape types;
- T is a finite non-empty set of terminal shape types, with $N \cap T = \varnothing$; (it corresponds to the set B introduced in the previous subsection);
- POS is a finite set of binary spatial and temporal relation identifiers, with $POS \cap N = \varnothing$ and $POS \cap T = \varnothing$;
- $S \in N$ denotes the starting non-terminal shape type;
- P is a finite non-empty set of *productions* of the following format:

$$(A_\Gamma \longrightarrow x_1(p_1)\mathbf{R_1}x_2(p_2)\mathbf{R_2}\ldots x_{m-1}(p_{m-1})\mathbf{R_{m-1}}x_m(p_m), Action)$$

where

  - A is a non-terminal shape type,
  - Γ is a set of triples $\{(NT_j, Cond_j, \Delta_j)\}_{j=1\ldots k}$, $k \geq 0$, used to dynamically insert new terminal shape in the input during the parsing process, enhancing the expressive power of the formalism. In particular, NT_j is a terminal shape type to be inserted in the input sentence; $Cond_j$ is a pre-condition to be verified in order to insert NT_j; Δ_j is the rule used to compute the values of the attributes of NT_j from those of $x_1, \ldots, x_m$.

- $x_1(p_1)\mathbf{R_1}x_2(p_2)\mathbf{R_2}\ldots x_{m-1}(p_{m-1})\mathbf{R_{m-1}}x_m(p_m)$ is a linear representation with respect to POS where each x_i is a shape type in $N \cup T$, each p_i is an optional value, named discriminant value, between 0 and 100 indicating the relevance of shape x_i in the modeled symbol, and each $\mathbf{R_j}$ is a sequence $\langle REL_{j_1}^{h_1}(t_1), \ldots, REL_{j_k}^{h_k}(t_k)\rangle; \langle REL_{j_{k+1}}^{h_{k+1}}(t_{k+1}), \ldots, REL_{j_n}^{h_n}(t_n)\rangle$ with $1 \leq k \leq n$. Each $REL_{j_i}^{h_i}(t_i)$ relates attributes of x_{j+1} with attributes of x_{j-h_i}, with $0 \leq h_i < j$, by means of a threshold t_i. Notice that we denote $REL_1^0(0)$ simply as REL_1. Each $REL_{j_i}^{h_i}(t_i)$ may also be a temporal relation, in this case t_i represents the time interval value that relates the two shape types.
- Action specifies the actions that have to be executed when the production is reduced during the parsing process. The actions are enclosed into the brackets { }.

A SkG sentence combines symbols with spatial and temporal relations. As we will show in the following section, the discriminant values associated to the shapes of the productions help to recognize messy or incomplete symbols. Indeed, high discriminant values are associated to the shapes that can be used tp distinguish a symbol from the others with a high degree of certainty. As an example, class diagrams include Aggregation and Composition symbols, which have a very similar shape, as shown in Figure 3. When these symbols are sketched, the strokes composing the Aggregation symbol are also part of the Composition symbol as depicted in Figure 5a. As a consequence, the shapes of the Composition production that allow to distinguish a Composition from an Aggregation, i.e., the strokes that fill the diamond, will have associated a high discriminant value.

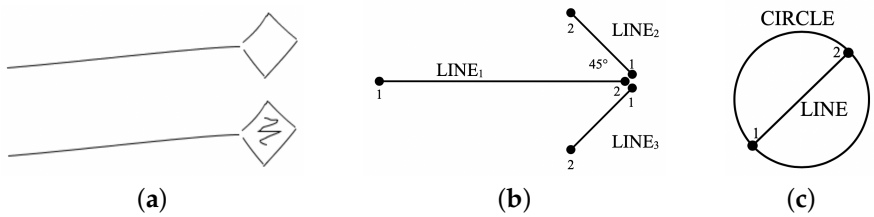


Figure 5. A sketched version of Aggregation and Composition symbols (**a**), Association symbol (**b**), and a temporal gesture (**c**) as described by the SkG grammar productions.

The following production specifies the *Association* symbol of class diagrams:

(Association → $\text{LINE}_1(60)$ ***<joint$_{2_1}$(t_1), rotate(45, t_2)>*** $\text{LINE}_2(20)$ ***<joint$_{2_1}^1$(t_1), rotate^1(-45,t_3)>*** $\text{LINE}_3(20)$,
{Association.attach(1) = LINE_1.attach(1); Association.attach(2) = LINE_1.attach(2) ∪ LINE_2.attach(1) ∪ LINE_3.attach(1);})

The Association symbol is composed by three lines, as shown in Figure 5b (the attributes are represented with bullets). The attribute 2 of LINE_1 is joined to the attributes 1 of LINE_2 and LINE_3 (due to relations $joint_{2\_1}$ and $joint_{2\_1}^1$, respectively), and the latter are rotated with respect to the former of 45 and −45 degrees, respectively. The values t_1, t_2, and t_3 specify the error margins in the satisfaction of the relations. Attributes 1 and 2 of Association are calculated from the attribute values of the three lines. Finally, the discriminant values indicate that the shaft of the Association symbol has a greater weight with respect to the other segments for discriminating an incomplete arrow symbol.

The following productions specify the Class, Composition, and Aggregation symbols of the class diagrams:

1. (Class → Rectangle,
   {Class.attach(1) = Rectangle.attach(1); })
2. (Class → Rectangle <$contain(t_2)$, $joint_{1\_1}(t_1)$, $joint_{1\_2}(t_1)$> LINE(40) <$parallel(t_4)$> LINE
   {Class.attach(1) = Rectangle.attach(1); })
3. Rectangle → $LINE_1$(15) <$joint_{2\_1}(t_1)$, $rotate(90, t_3)$>
   $LINE_2$(15) <$joint_{2\_1}(t_1)$, $rotate(90, t_3)$>
   $LINE_3$(15) <$joint_{2\_1}(t_1)$, $rotate(90, t_3)$, $joint_{1\_2}^2(t_1)$> $LINE_4$(15),
   {Rectangle.attach(1) = $LINE_1$.border ∪ $LINE_2$.border ∪ $LINE_3$.border ∪ $LINE_4$.border; })

1. (Composition → LINE(30) <$joint_{2\_1}(t_1)$ > Head,
   {Compostion.attach(1) = LINE.attach(1); Composition.attach(2) = Head.attach(2);})
2. (Head → Diamond <$contain(t_2)$> Scribble(20) <$contain^1(t_2)$> $Scribble_1$(20), {Head = Diamond;})
3. (Head → $Head_1$ <$contain(t_2)$> Scribble, {Head = $Head_1$;})
4. (Diamond → $LINE_1$(10) <$joint_{2\_1}(t_1)$, $rotate(90, t_3)$>
   $LINE_2$(5) <$joint_{2\_1}(t_1)$, $rotate(90, t_3)$>
   $LINE_3$(5) <$joint_{2\_1}(t_1)$, $rotate(90, t_3)$, $joint_{1\_2}^2(t_1)$> $LINE_4$(10),
   { Diamond.attach(1) = $LINE_1$.attach(1); Diamond.attach(2) = $LINE_3$.attach(1);
   Diamond.contain = Rectangle($LINE_1$.attach(1), $LINE_2$.attach(1), $LINE_3$.attach(1), $LINE_4$.attach(1));})
5. (Scribble → LINE)
6. (Scribble → ARC)

1. (Aggregation → LINE(40) <$joint_{2\_1}(t_1)$ > Diamond,
   { Aggregation.attach(1) = LINE.attach(1); Aggregation.attach(2) = Diamond.attach(2);})
2. (Diamond → $LINE_1$(20) < $joint_{2\_1}(t_1)$, $rotate(90, t_3)$> $LINE_2$(10) < $joint_{2\_1}(t_1)$, $rotate(135, t_3)$>
   $LINE_3$(10) < $joint_{2\_1}(t_1)$, $rotate(135, t_3)$, $joint_{1\_2}^5(t_1)$ > $LINE_4$(20),
   {Diamond.attach(1) = $LINE_1$.attach(1); Diamond.attach(2) = $LINE_3$.attach(1);
   Diamond.contain = Rectangle($LINE_1$.attach(1), $LINE_2$.attach(1), $LINE_3$.attach(1), $LINE_4$.attach(1));})

The Composition symbol is composed of a line joined with a diamond shape containing a scribble of at least two primitive shapes, whereas the Aggregation is a line joined with a diamond shape. Thus, the existence of the scribble allows us to distinguish a Composition from an Aggregation symbol. For this reason, the discriminant values associated to the strokes forming the scribble of the Composition symbol have a weight greater than the other shapes.

Temporal relations are particularly useful for defining multi-stroke editing gestures such as deleting, moving, copying, because they constrain users to draw the sequence of strokes forming such gestures within a fixed time. As an example, the following production defines an editing gesture formed by a circle with a line inside of it, which logically selects the set of strokes inside the circle.

(Select → CIRCLE <$contain(t_1)$, $joint_{1\_1}(t_2)$, $joint_{1\_2}(t_2)$, ***before*(t_3)**> LINE,
{ SET selection = CIRCLE.contain – LINE;
sketchInterface.highlight = selection;
sketchInterface.remove(CIRCLE);
sketchInterface.remove(LINE);})

Notice that the line must be drawn within t_3 milliseconds after the circle has been drawn. The Selection gesture is composed by a circle and a line, as shown in Figure 5c. The action associated to the production computes the strokes of the drawings which are contained in the circle gesture except the line gesture. These strokes are highlighted and then the circle and line gestures are removed from the sketch interface.

SkG grammars allow to specify both the symbol grammars, which define the graphical representation of symbol shapes of a language as geometric compositions of primitive shape, and the language grammars, which specify the possible sentences of the language as composition of the shapes defined by symbol grammars through spatial relations. For example, the previous grammar describing the Association symbol is used to recognize the ASSOCIATION links of class diagrams, whereas the following productions define the complete language grammar for UML class diagrams:

1. (ClassDiagram → Iterator)
2. (Iterator → CLASS, {Iterator.attach(1) = CLASS.attach(1);})
3. (PIterator → PACKAGE, {PIterator.contain = PACKAGE.contain;})
4. (Iterator $_{\{(PLACEHOLD;\ true;\ PLACEHOLD.attach(1) = Node.attach(1) - Link.attach(1))\}}$ → Iterator$_1$ <$joint_{1\_2}(t_1)$>;<$\overline{joint_{1\_1}(t_2)}$> Link <$joint_{1\_1}(t_1)$> Node, {Iterator.attach(1) = Iterator$_1$.attach(1) – Link.attach(2);})
5. (Iterator $_{\{(PLACEHOLD;\ true;\ PLACEHOLD.attach(1) = Node.attach(1) - Link.attach(2))\}}$ → Iterator$_1$ <$joint_{1\_1}(t_1)$>;<$\overline{joint_{1\_2}(t_2)}$> Link <$joint_{2\_1}(t_1)$> Node, {Iterator.attach(1) = Iterator$_1$.attach(1) – Link.attach(1);})
6. (Iterator → Iterator$_1$ <$joint_{1\_1}(t_1)$, $joint_{1\_2}(t_2)$> Link, {Iterator.attach(1) = (Iterator$_1$.attach(1) – Link.attach(1)) – Link.attach(2) ;})
7. (Iterator → PIterator$_1$ <$contain(t_3)$> Node, {Iterator.attach(1) = Node.attach(1);})
8. (Iterator → PIterator$_1$ <*any*> PNode, {Iterator.attach(1) = PNode.attach(1);})
9. (Iterator → Iterator$_1$ <*any*> Node, {Iterator.attach(1) = Node.attach(1);})
10. (Iterator → Iterator$_1$ <*any*> PNode, {Iterator.contain = PNode.contain;})
11. (Node → CLASS, {Node.attach(1) = CLASS.attach(1);})
12. (Node → PLACEHOLD, {Node.attach(1) = PLACEHOLD.attach(1); })
13. (PNode → PACKAGE, {PNode.contain = PACKAGE.contain;})
14. (Link → ASSOCIATION, {Link.attach(1) = ASSOCIATION.attach(1); Link.attach(2) = ASSOCIATION.attach(2);})
15. (Link → AGGREGATION, {Link.attach(1) = AGGREGATION.attach(1); Link.attach(2) = AGGREGATION.attach(2);})
16. (Link → INHERITANCE, {Link.attach(1) = INHERITANCE.attach(1); Link.attach(2) = INHERITANCE.attach(2);})
17. (Link → COMPOSITION, {Link.attach(1) = COMPOSITION.attach(1); Link.attach(2) = COMPOSITION.attach(2);})

This grammar defines a class diagram as an Iterator (production 1), which describes a set of classes related through Link symbols (productions 2, 4, 5, and 6), and eventually packages containing them (productions 3, 7). Iterator is recursive and might be not connected through Link (productions 8, 9, and 10).

6. The Sketch Recognition System

In the following, we describe the proposed sketch recognition system. The description of the system will mainly focus on the online recognition, whereas the application of the proposed approach to the offline case is discussed at the end of the section. The lower layer recognizer (see Figure 2) is invoked upon stroke editing, whereas the recognition result is provided by the higher level recognizer to the sketch editor on demand.

6.1. The Primitive Shape Recognizer

As shown in Figure 2, the primitive shape recognizer aims to classify the stroke pixels into primitive geometric objects, such as lines, arcs, ellipses, and so on. Figure 6 shows the three phases of the primitive shape recognition process.

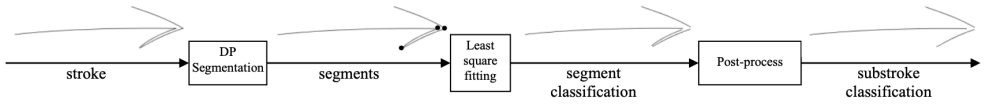


Figure 6. The three phases of the primitive shape recognition process.

The strokes are suitably split into single-stroke segments aiming to recognize multi-stroke symbols. Segmentation is a well-studied problem and powerful approaches have been presented in the literature, see, e.g., [53,54]. In the current implementation of the proposed approach, the segmentation is accomplished by the algorithm that is proposed in [55]. In particular, the algorithm splits a free-hand stroke into line segments and elliptical arcs by using a dynamic programming technique. The generated segments are classified by using suitable fitting functions: least-square fitting to fit a fragment into a line segment [56] or an elliptical arc [57]. The post-process phase aims to eliminate possible redundancies, such as overtracing, by merging primitive shapes satisfying particular conditions.

The output of the primitive shape recognizer on a shape S is a set of interpretations for each segmented stroke identified in S. This information is stored in a Dictionary.

6.2. The Symbol Recognizer

Symbol recognizers cluster the segment classifications produced by the primitive shape recognizer into symbols of the domain language. Each recognizer is automatically generated by the parser generator starting from a symbol grammar and uses a parsing technique that extends the approaches that were proposed in [58,59]. In particular, the recognizers scan the input in an incremental and non-sequential way, driven by the spatial relations specified in the grammar productions. Figure 7 shows the architecture of a symbol recognizer.

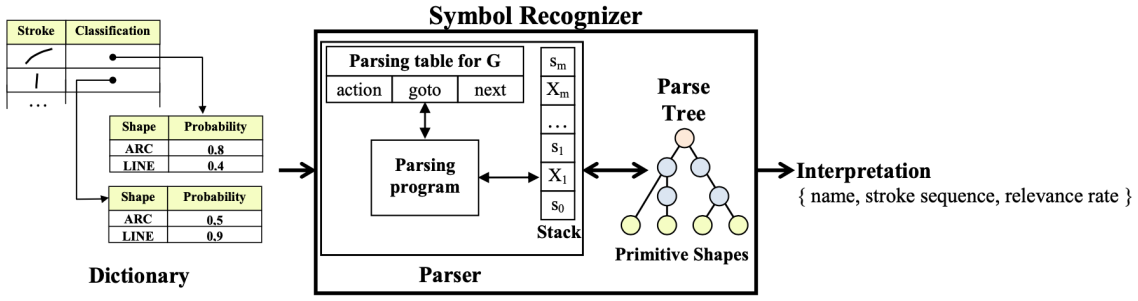


Figure 7. The architecture of a symbol recognizer.

The parsing table for a symbol grammar G is built by giving in input G to the parser generator. The incremental parser takes in input:

- the new shapes identified by the primitive shape recognizer, which are saved in the Dictionary;
- a parse tree, which represents the recognized strokes; and,
- an internal stack, which is built on the input analyzed so far.

The parsing program updates the parse tree and the stack based on the new strokes. The nodes of the tree have associated the discriminant rate of the strokes associated to the leaves of its subtree. The symbol recognizer outputs the name of the symbol, the stroke sequence analyzed by the parser, and the discriminant rate.

The parsing table, generated from a grammar, is composed of a set of rows divided into three sections: Action, Goto, and Next. Each row is composed of a set of one or more sub-rows each corresponding to a parser state. The Action and Goto sections are similar to the ones used in the LR parsing tables for string languages [20], while the Next section is used by the parser to select the next symbol to be processed. An entry Next[k] for a state k contains the triple (relations, x, DV), which drives the parser in selecting a symbol x with a discriminant value DV satisfying relations [60]. The special entry (start, S) is used to retrieve the first symbol to be parsed.

As an example, Table 1 shows the parsing table for the Composition grammar. Note that the special next entry (any, *) returns any terminal present in the input.

Table 1. Parsing table for Composition grammar.

State		Action		Goto				Next
		LINE	ARC	Composition	Head	Diamond	Scribble	
0		:sh1		:10				(*start*, LINE, 30)
1		:sh2			:3	:4		($joint_{2\_1}(t_1)$, LINE, 10)
2		:sh5						($joint_{2\_1}(t_1)$, *rotate*(90,t_3), LINE, 5)
3	1	:sh_r5					:r3	(*contain*(t_2), LINE)
	2		:sh_r5				:r3	(*contain*(t_2), ARC)
	3	r1						(*any*, $\star$)
4	1	:sh_r5					:6	(*contain*(t_2), LINE, 20)
	2		:sh_r5				:6	(*contain*(t_2), ARC, 20)
5		:sh7						($joint_{2\_1}(t_1)$, *rotate*(90,t_3), LINE, 5)
6		:sh_r6					:r2	($contain^1(t_2)$, LINE, 20)
			:sh_r6				:r2	($contain^1(t_2)$, ARC, 20)
7		:sh_r4						($joint_{2\_1}(t_1)$, *rotate*(90,t_3), $joint^2_{1\_2}(t_1)$, LINE, 10)
10		accept						–

Note: sh=shift, r=reduce, sh_r=shift reduce.

The parsing algorithm associated to a symbol recognizer is described in Appendix A. Figure 8 shows the recognition performed by the aggregation symbol recognizer generated from the productions that are described in the previous section.

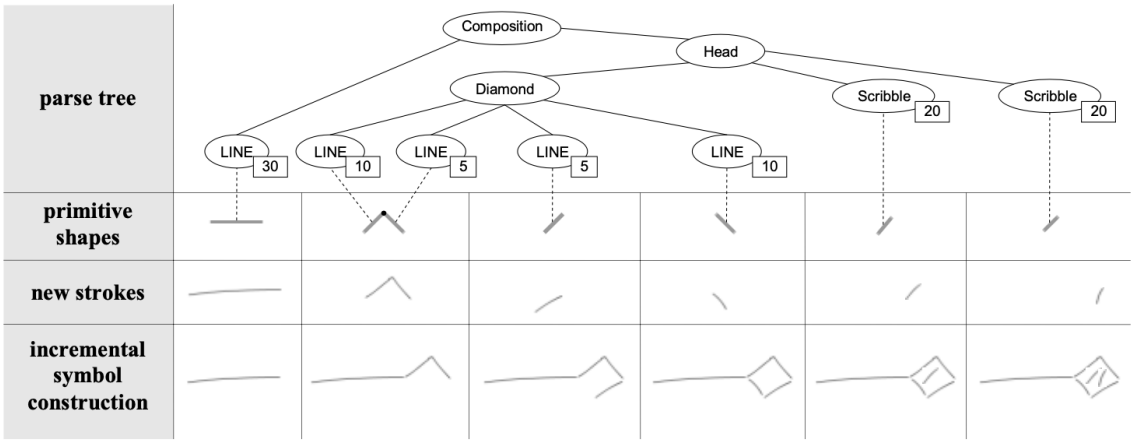


Figure 8. Incremental recognition of a Composition symbol.

6.2.1. Improving Recognition Robustness with Error Recovery Techniques

In this section we extend the parsing process with error recovery techniques to cope with the intrinsic variability of hand-drawn sketches, e.g., missing symbol strokes or strokes that are difficult to identify [61]. In particular, in the case of a missing or misrecognized symbol, the parsing algorithm actives a procedure to proceed in the recognition of the incomplete symbol. Obviously, the procedure should not be applied if many strokes of the symbol are still missing. To this end, we use the discriminant values that are associated to the partially recognized symbols both to disambiguate the recognition of similar symbols and limit the number of missing strokes for a symbol. If t is the threshold value then a symbol S is (partially or completely) recognized when dv, the sum of discriminant values associated to the recognized strokes of S, is greater than t. In case dv exceeds $100 - t$, the error recovery terminates, since the recognized symbol will never exceed threshold t, even if all the remaining symbol strokes have been drawn. In the following, we provide the algorithms implementing the error recovery technique.

```
Recovery(){
    PT = parsingtable[parser];
    state = stack[parser].currentState;
    while (state != null) {
        (r, s, imp_value) = PT.next[state];  //for multiple instances the triple with lowest discriminant value is selected
        if (imp_value > 100-threshold[parser])
            exit;
        threshold[parser] = threshold[parser]+ imp_value;
        newStroke = Fit(r, s);
        state = ContinueParsing(newStroke);
    }
}

Fit(r, s) {
    rep = repository[parser];
    foreach(x in rep) {
        if (r is a valid relation for x) {  //using complete classification
            delete x from rep;
            return x;
        }
    }
    simulatedStroke = use the constraint solver to calculate a shape of s type compatible with the stack and relation r;
    return simulatedStroke;
}

ContinueParsing(s) {
    action[parser].shift(s);
    rep = repository[parser];
    input[parser] = rep;
    action[parser].continue;   //reactivate the parser on rep
    input[parser] = PrimitiveShapeRecognizer;   //restore the input
    if(state[parser] != accept and rep.empty == true)
        return null;
    if (state[parser] == accept)
        return null;
    return stack[parser].currentState;
}
```

A symbol recognizer automatically invokes the Recovery function when a syntax error occurs. In this case, if the symbol can still be recognized the Fit function looks for a stroke able to reactivate the parser, whereas the *ContinueParsing* function continues the parsing from the stroke following the missing stroke.

The Fit function first tries to find a stroke s to fix the syntax error from the repository of unmatched strokes. If it is not found, the function creates a dummy stroke simulating s. In both cases, the output stroke is used by *ContinueParsing* function for updating the parsing state and then reactivating the parser.

After the recovery process, the symbol recognizer checks the acceptance state of the parser to verify whether the symbol is completely or partially recognized.

As an example, let us suppose that the user has drawn the Composition symbol that is shown in Figure 9a which has a missing stroke. It is composed of two strokes s_1 and s_2 which are segmented into 7 simple strokes as shown in Figure 9b. Moreover, let us suppose that the recognizer has a threshold value of 80 for the recognition of Composition symbols.

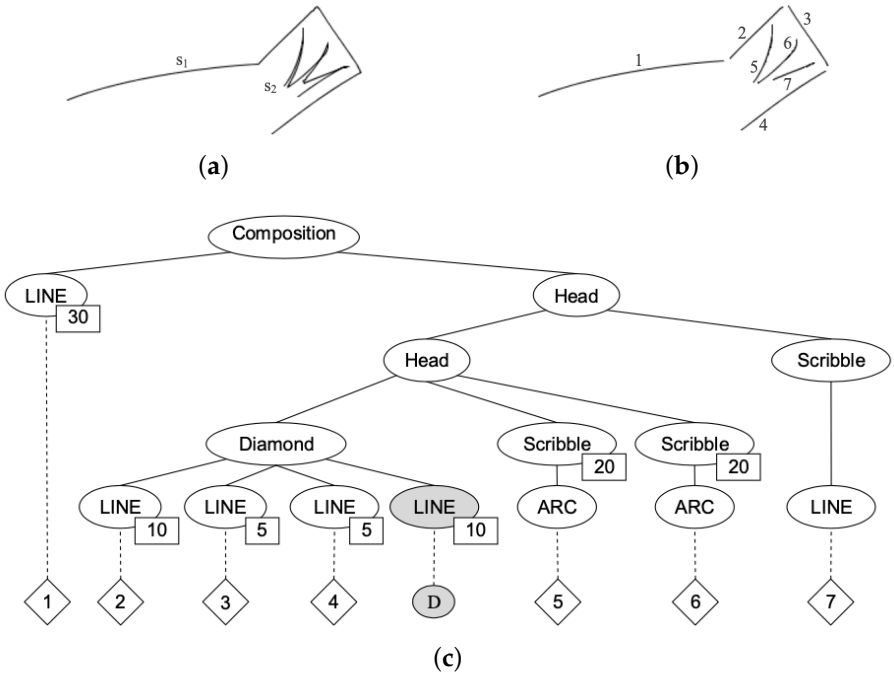


Figure 9. A Composition symbol with a missing stroke (**a**), its segmentation (**b**), and the parse tree constructed by the parser with error recovery (**c**).

After the recognition of strokes labeled from 1 to 4 the Composition parser reaches state 7, but then fails because the next symbol in the input is missing. The parser invokes the error recovery procedure that uses the triple ($joint_{2\_1}(t_1)$, $rotate(90, t_3)$, $joint^{2}_{1\_2}(t_1)$, LINE, 10) to check whether the discriminant value 10 exceeds 20 (i.e., 100 less the threshold). Then, it looks for a valid stroke compatible with the previous triple in the repository. Since it is not found Fit function simulates the line completing the diamond shape (which is depicted in gray in Figure 9c) and the parser is reactivated. In particular, the parser moves from state 7 to state 4, and completes the recognition of the remaining strokes reaching the acceptance state.

6.3. The Language Recognizer

The language recognizer takes in input the results that are produced by the symbol recognizers and exploits contextual information for determining the best possible interpretation. This includes the recognition of incomplete or inaccurate symbols, and the pruning of unnecessary active symbol recognizers.

When a new stroke is drawn by the user, all of the active symbol recognizers receive in input result of the primitive recognizer. The strokes that do not compound with those previously parsed are temporarily stored in the unmatched stroke repository of its recognizer. Instead, the recognizers that are able to parse the stroke provide their output to the language recognizer. The latter is implemented as a non-deterministic parser automatically generated from the language grammar [59]. Such a parser scans the results of the symbol recognizers and selects the most suitable interpretation according to the discriminant rates. When the interpretation of a symbol S exceeds a predefined discriminant threshold, the language recognizer discards the active recognizers that have analyzed strokes in S.

Figure 10 depicts how a class diagram is recognized during the editing process. The figure also shows the results of the active symbol recognizers and highlights the interpretation selected by the language recognizer.

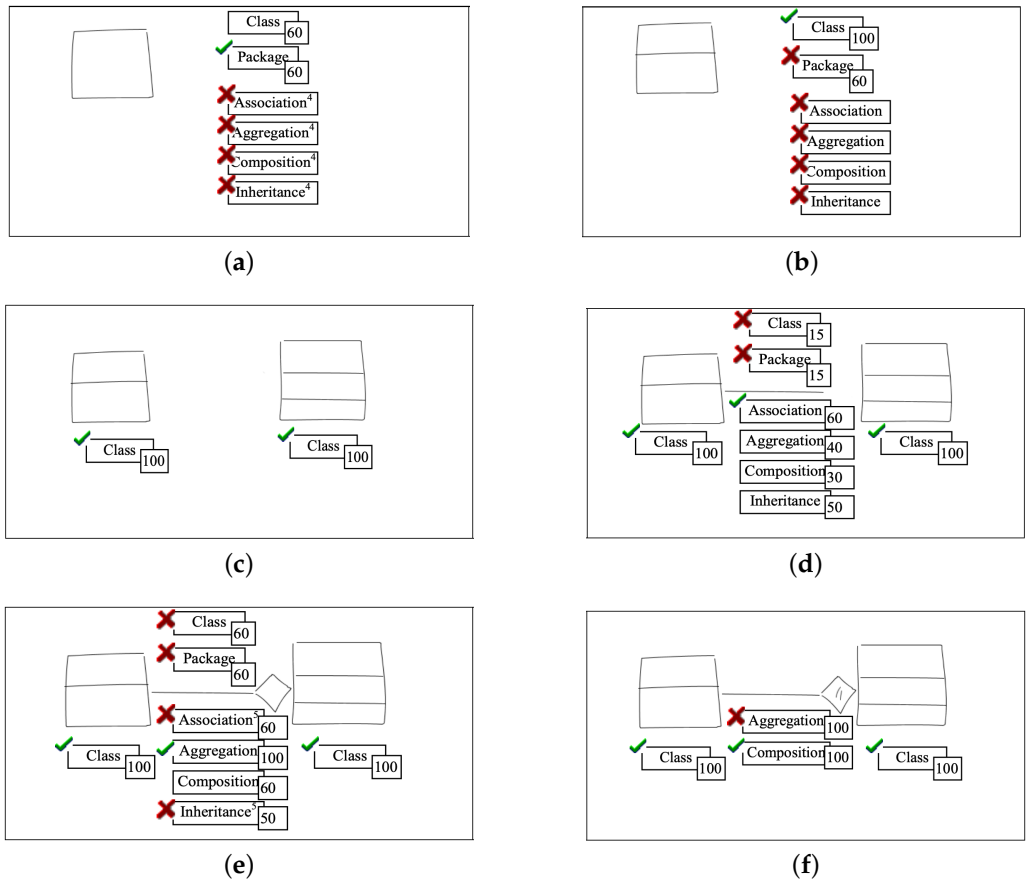


Figure 10. The incremental recognition of a class diagram. (**a**) The square is parsed by Class and Package recognizers with a discriminant rate of 60. (**b**) Only the Class recognizer is able to compound the line within the square to the previously recognized strokes. (**c**) The same recognition process is applied to the second Class symbol. (**d**) The language recognizer interprets the line between the two class symbols as Association since the corresponding recognizer obtains the highest discriminant value. (**e**) The rhombus near the line is compounded to the previous stroke by Composition and Aggregation recognizers. (**f**) The strokes drawn inside the rhombus lead to the recognition of a Composition symbol.

In particular, the selected interpretations are indicated with a checkmark, whereas those pruned with a cross-mark. The user starts by sketching the square of a class symbol with a single stroke, as shown in Figure 10a. The stroke is segmented into four lines and the symbol recognizers able to parse all of the strokes are the Class recognizer and the Package recognizer, both with a discriminant rate of 60. The other recognizers are able to parse a single line stroke. Since the lines composing a square have a low probability to be connector symbols (i.e., Association, Aggregation, Composition, and Inheritance) the language recognizer prunes all of the the active recognizers of connector symbols.

Successively, the user draws with a single stroke a line within the square, as shown in Figure 10b. Only the Class recognizer is able to compound the line to the previously recognized strokes. Moreover, other recognizers are launched on the new stroke. Because the class recognizer reaches a discriminant rate of 100, the language recognizer discards the remaining active recognizers. The same recognition process is performed on a second sketched class symbol, as shown in Figure 10c.

When the user draws a line between the two class symbols (see Figure 10d), a new recognizer of each domain symbol is activated. The language recognizer prunes class and package recognizers, since only connector symbols can be drawn between class symbols. Moreover, it selects the Association interpretation which has the highest discriminant value. By drawing a rhombus near the line symbol, as shown in Figure 10e several new recognizers are activated. Only the Composition and Aggregation recognizers are able to compound the rhombus to the previously recognized strokes. The language

recognizer prunes the Class and Package recognizers because neither classes nor packages can be related to a class symbol, and the Association and Inheritance recognizers, because they parse a subset of the strokes that are used to recognize the aggregation and composition symbols. It is worth to note that the Composition recognizer is not discarded because a composition symbol can be drawn starting from an Aggregation symbol. Indeed, in Figure 10f, the strokes that are drawn inside the rhombus lead to the recognition of a Composition symbol. In this case, the language recognizer prunes all of the recognizers activated on the strokes of the Composition symbol. In particular, the Aggregation recognizer is discarded, because its recognized symbol strokes are included in the composition symbol.

The language recognizer is also able to establish whether a class diagram is syntactically correct by using the embedded parser. As an example, only the class diagram of Figure 10d is not syntactically correct, because the association symbol is linked to one class only.

6.4. Offline Sketch Recognition

The proposed recognition approach can also be applied to offline sketch recognition. The layers of the system that have to be adapted to this recognition mode are the primitive shape recognizer and the symbol recognizers.

In the offline case, the input to the system is a bitmap image that is produced by image acquisition devices. Thus, the first task of the sketch recognition system is the analysis of the input image and its conversion into a set of primitive shapes. Because the scanning process can introduce noise into the sampled image bitmap of the original sketch, an image filter can be used to preprocess the image bitmap. The output of the filter is a smoothed image bitmap that is presented to the segmentation step, which identifies a set of primitive shapes that are used by the individual recognizers. The set of shapes is sorted according to a topological order.

The symbol recognizers receive the sequence of primitive shapes from the low-level recognizer and apply their parser for identifying the domain symbols. Notice that, in the offline case, the generated parsers ignore the temporal information provided in the symbol grammars.

7. Recognition Accuracy Evaluation and Usability Study

We have implemented the proposed recognition model in *SketchBench* [58], a system that provides sketch recognition capabilities for pen-based applications. This system has been used to evaluate the effectiveness of the proposed recognition system. In particular, in the domain of UML class diagrams we have run a user study for analyzing recognition performances and a usability study to highlight the advantages of *SketchBench* over traditional WIMP (Windows, Icons, Menus, and Pointing) CASE tools.

7.1. Uml Class Diagram Recognition Study

The goal of this user study is to evaluate the recognition performances of the proposed approach and compare them against the results that were obtained with a previous recognition system [58].

7.1.1. Experiment Setup

We recruited twenty subjects with little experience using sketch-based interfaces and with basic knowledge of UML diagrams. They took a lesson of 10 min. to learn the principles of drawing by sketches and introduced them the main *SketchBench* features, included the multi-stroke symbol recognition capabilities.

We asked to the subjects to use the system for some minutes until they felt comfortable with it and with the pen-based input device. Subsequently, we gave them the five class diagrams (in the following indicated with $D1, \ldots, D5$) depicted in Appendix B, which they had to draw in an unconstrained fashion using *SketchBench*. We collected one hundred sketches of varying complexity containing between 22 and 94 strokes and between five and 17 symbols. The subjects knew that they were not being timed and that they were able to erase strokes during the editing process only by using the undo

operation. Moreover, *SketchBench* does not provide feedback to the users during the editing process in order to not interfere with data collection and avoid any kind of adaptation from the user.

Because the direct comparison with previous work is difficult, to measure the benefits of our approach, we use as a baseline the bottom-up approach presented in [58]. This approach combined low-level shapes into language symbols using a single monolithic LR-based parser for the domain language. The interpretation of the sentence was done by selecting the symbol interpretations that cover the higher number of strokes with a reasonable percentage of accuracy.

7.1.2. Results and Discussion

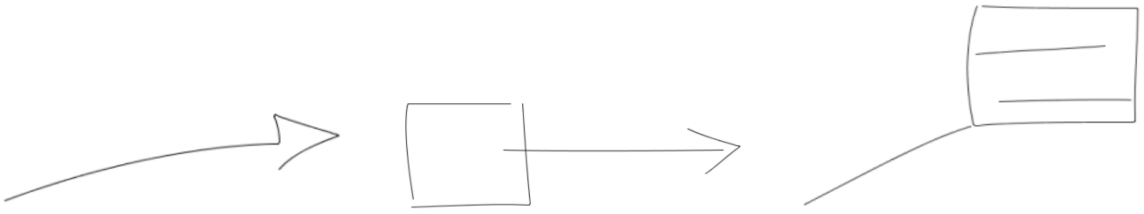
The recognition performance have been measured by computing the number of correctly identified symbols in each diagram. We have used the precision and recall metrics to measure the recognition rates [62]. Precision is the ratio of the actual recognized symbols over the total recognized symbols, whereas recall is the ratio of the number of recognized symbols over the number of real symbols in the diagrams. Thus, a value that is equal to 1 for recall (for precision, respectively) means that all of the real symbols have been recognized (all the recognized symbols are real symbols, respectively).

The confusion matrix shown in Table 2 reports the recognition errors of the system. Rows contain the interpretations obtained for the drawn symbols, while columns contain the instances (mis-)recognized for a given symbol. For instance, the Package row indicates that 78 out of 86 Package symbols have been correctly recognized, seven have been misrecognized as Association, and one as Aggregation. Whereas, the Class column indicates that 455 Class symbol instances have been recognized, three of should be recognized as Aggregation (false positives). The diagonal reports the number of symbols recognized correctly.

Table 2. Confusion matrix for the class diagram symbols. Each row shows the number of symbols of a given type that were assigned to each class. The last column shows the recall and the last row the precision.

Shapes		Recognized						Recall
		Class	**Package**	**Association**	**Aggregation**	**Composition**	**Inheritance**	
Drawn	**Class**	**455**	5	22	9	2	1	0.95
	Package	0	**78**	7	1	0	0	0.98
	Association	0	0	**84**	9	2	8	0.84
	Aggregation	3	0	3	**51**	3	3	0.85
	Composition	0	0	2	8	**48**	2	0.80
	Inheritance	0	0	12	6	0	**142**	0.89
Precision		0.99	0.94	0.65	0.68	0.91	0.92	

The matrix highlights that many mistakes are due to confusing different views of the same structure, e.g., Associations that are recognized as Aggregations and Inheritances. As shown in Figure 11a, this might happen when the user draws sloppy arrows with a single stroke. In this case, the parser cannot solve the ambiguities due to missing contextual information. This also happens during the recognition of Aggregation, Composition, and Inheritance symbols.



(**a**) (**b**) (**c**)

Figure 11. Examples of sketched symbols misrecognized by the proposed approach. A sloppy arrow with a single stroke (**a**); an association symbol positioned on the corner of a messy Class symbol (**b**); an Association and a Class symbol misrecognized as a Composition (**c**).

The symbols with highest recognition rate are Classes and Packages. This depends on their unambiguous shape features, and on the context specified in the language grammar. In 4 cases a Class symbol has been misrecognized as a set of Association symbols. This occurred when the Class symbol is drawn as a messy square, as shown in Figure 11b. In 11 cases, a Class symbol has been misrecognized as Aggregation or Composition. This happened when an association symbol is positioned on the corner of a messy Class symbol. For instance, the Association and the Class symbols in Figure 11c are misrecognized as a Composition. The Package is never confused as a Class symbol, since it is the only symbol containing other symbols. This allows the system to correctly solve the ambiguities with the contextual information.

Figure 12 shows how the precision and recall values vary on the different diagrams. We can notice that no dependence exists between diagram sizes and precision and recall measures. This is mainly due to the independence of symbol recognizer execution, e.g, when a parser fails the performances of the other parsers are not affected considerably. The worst performances obtained on diagram 3 are mainly due to the poor contextual information that is available for ambiguity resolution.

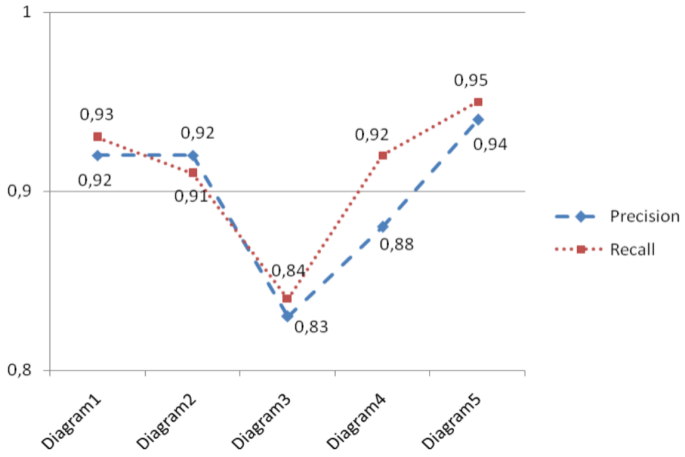


Figure 12. Precision and recall by diagram.

Tables 3 and 4 provide the recognition rates of the baseline (BL) and multi-layer (ML) algorithms. The results show that ML outperforms BL. In fact, BL correctly recognizes 79% of the symbols on average, whereas ML reaches 90% of correctly identified symbols. Only for Class symbols the recognition improvement obtained with ML is lower, this is due to the fact that the BL system already obtained good performances.

Table 3. Recognition rates for the baseline system (BL) and the proposed algorithm (ML). The size column indicates the number of strokes in each sketch.

	Size	#Symbols	Precision		Recall	
			BL	ML	BL	ML
D1	25	5	0.84	0.92	0.81	0.93
D2	35	6	0.79	0.92	0.77	0.91
D3	50	9	0.71	0.83	0.69	0.84
D4	60	10	0.74	0.88	0.74	0.92
D5	90	17	0.84	0.94	0.83	0.95
Average	52	9.4	0.79	0.90	0.77	0.89

Table 4. Recognition rates by shape.

	Total	Precision		Recall	
		BL	ML	BL	ML
Class	480	0.98	0.99	0.83	0.95
Package	80	0.72	0.94	0.78	0.98
Association	100	0.44	0.65	0.75	0.84
Aggregation	60	0.52	0.68	0.70	0.85
Composition	60	0.77	0.91	0.55	0.80
Inheritance	160	0.85	0.92	0.73	0.89

7.2. Sketchbench Usability Study

This second user study has the goal of analyzing how *SketchBench* is easy to use for drawing class diagrams with respect to a traditional WIMP UML CASE tool.

7.2.1. Experiment Setup

The usability study of *SketchBench* has been performed by recruiting sixteen undergraduate students from a software engineering class, which were not involved in the recognition study. All of the students were familiar with UML class diagram notation, but they had no experience with the use of CASE (Computer Automated Software Engineering) tools and sketch-based interfaces. They took a lesson of 15 min to learn the principles of drawing by sketches. Moreover, we introduced them to the main *SketchBench* features, included the multi-stroke symbol recognition capabilities, and we show them how to sketch class diagrams. Notice that the version of *SketchBench* used in this study supported the deletion of any stroke during the editing process. Moreover, they took a lesson of 15 min. to learn the main features of *ArgoUML* for editing class diagrams.

The students were divided in two groups of eight subjects (A and B) and each student performed two tasks (*t1* and *t2*). Group A used ArgoUML to solve *t1* and *SketchBench* for *t2*, whereas group B solves *t1* with *SketchBench* and *t2* with ArgoUML. Each task consisted of creating a class diagram for a hotel reservation software according to a given use case. In particular, the use case for task *t1* was "the reservation of a room", whereas the use case for task t2 was "check-out of a room". For each task, the students had twenty minutes to solve it by using the assigned tool only.

The authors observed how students worked with the tools during the study. After students finished each task, they completed the post-task questionnaire that is shown in Table 5a regarding their experience with the used tool. At the end of the experiment, the subjects filled the conclusive questionnaire shown in Table 5b to report their positive/negative opinions on the two tools. The judgment to the questions was given within a range. Indeed, the questions expect closed answers according to a Likert scale: from 1 (Very low) to 5 (Very high).

Notice that *SketchBench* does not yet support text recognition, thus to support the user in the definition of a class diagram, including the class names, field and method declarations, and so on, we have added on the interface a button for switching from drawing mode to text mode editing, and vice versa. The strokes edited during the text-editing mode were not sent to the recognizer.

The *SketchBench* user interface also provides the subjects a button for requiring the current interpretation of the sketch drawn so far. The interpretation was given by associating a label to each recognized symbol and by coloring the class symbols and the relationships differently. As an example, Figure 13 shows a class diagram sketched by a student for solving task *t1*. In particular, the recognized class symbols are visualized in green, the relationships are in blue and, to provide an unambiguous description of the symbol interpretation, all recognized symbols have annotated a text label.

Table 5. Questions regarding the usability of *SketchBench* (a) and questions regarding the subject experience in the study (b).

(**a**)	(**b**)
Questionnaire 1	**Questionnaire 2**
1.1 How difficult is it for you create a class diagram?	
1.2. How difficult is it for you to create a class object?	2.1. I prefer *SketchBench* over ArgoUML for constructing UML class diagrams.
1.3. How difficult is it for you to create an association between class objects?	2.2. Overall, *SketchBench* is easy to use for constructing a UML class diagram.
1.4. How difficult is it for you to create an aggregation association?	2.3. Overall, it is pleasant to use *SketchBench*.
1.5. How difficult is it for you to create a generalization association?	2.4. Using the *SketchBench* does not require a lot of mental effort wrt ArgoUML.
1.6. Each set of operations does not produce a predictable result?	2.5. What do you like least about the *SketchBench*?
1.7. Overall, how difficult is it for you to use the system?	2.6. What do you like most about this *SketchBench*?
1.8. How difficult is it for you find the solution to the proposed problem?	

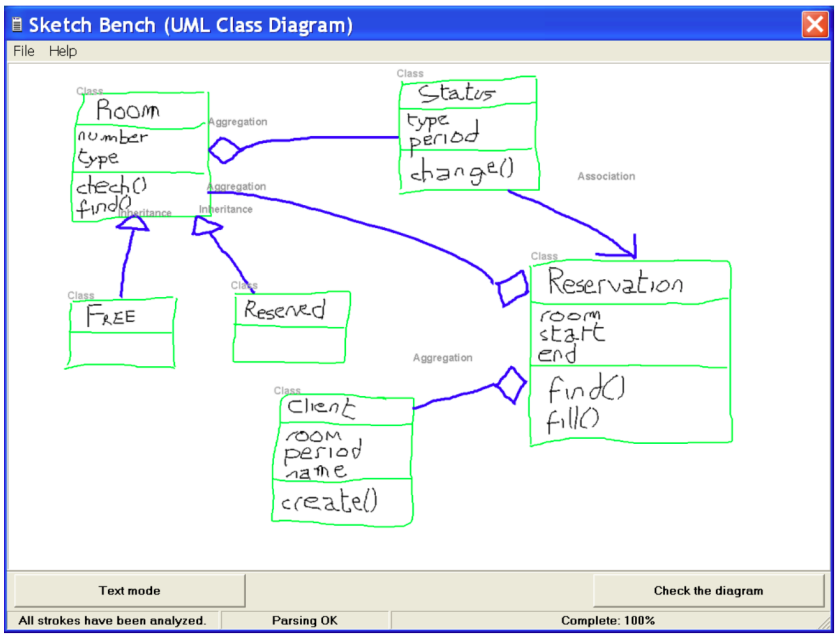


Figure 13. The visualization of the *SketchBench* interpretation for a user sketch.

7.2.2. Results and Discussion

Overall the assigned tasks were correctly completed by 9 students, 5 of group A and 4 of group B. Two students of group A failed to solve an assigned task: one using ArgoUML on task *t1* and one using *SketchBench* on task *t2*. In only one case a student did not complete the solution (for task *t2*) due to the inability of *SketchBench* to recognize a part of the student diagram. For group B, one student failed to solve both the tasks, two students failed to solve task *t2* using ArgoUML, and one student drew an unrecognizable sketch for task *t1*.

Boxplot in Figure 14 reports the distribution of the times required to solve a task by each group. The boxplot shows median values (horizontal lines), interquartile ranges (boxes), the largest and smallest observations (whiskers), and outliers (circles). Notice that label X_i in a plot indicates the data on group X for task i. We can observe that for task *t1 SketchBench* (used by group B) has considerably reduced the time to find the correct solution. This was mainly due to the ability of *SketchBench* interface to allow users to reason directly on the solution. The outlier in the boxplot is the time of the student, which created a sketch that *SketchBench* was not able to recognize.

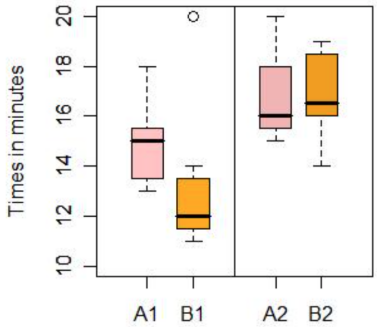
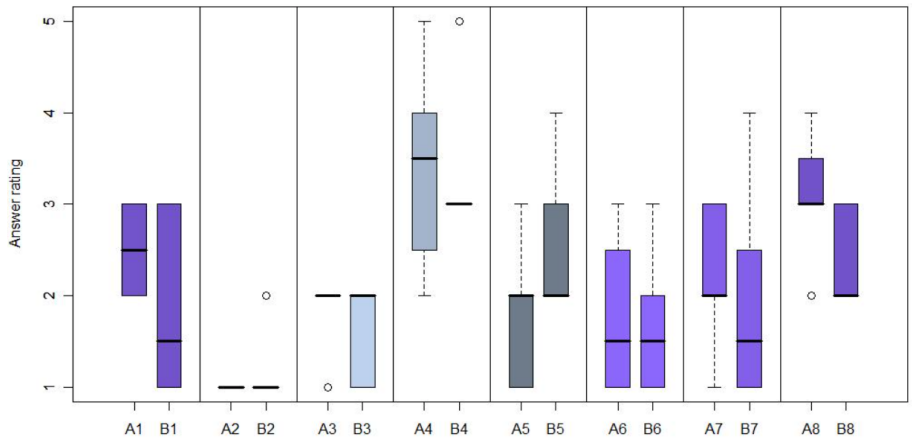


Figure 14. Boxplot showing the distribution of times to solve the two tasks by each group.

On the contrary, there is no significant time difference between the two tools for task *t2*. The advantages of *SketchBench* that are highlighted in *t1* are weakened by the increased complexity of the diagram to be drawn. Indeed, we observed that the students have encountered many difficulties in the modification of the sketches and in the editing of textual labels. Moreover, some ArgoUML facilities, such as the syntax-directed editing, used by the students of group A for solving *t1* were no longer available to solve *t2*.

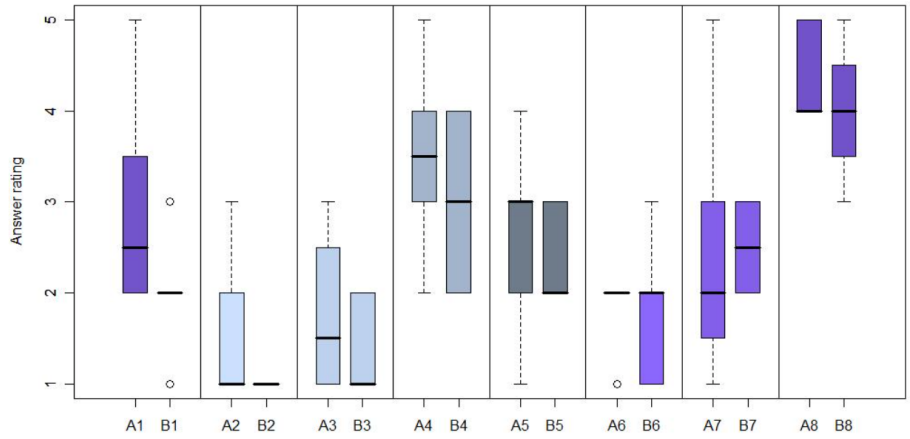
The boxplots presented in Figure 15 show the distributions of user answers to the post-task questionnaire. In particular, Figure 15a (Figure 15b, resp.) allows for comparing the answers of groups A and B to the questions on task *t1* (*t2*, resp.).

The ratings for question 1.1 reveal that the students consider *SketchBench* better than ArgoUML in the editing of simple class diagrams. In particular, for task *t1*, even if the maximum editing difficulty is the same for both tools, on average *SketchBench* is better. For task *t2*, whose solution requires more symbols than *t1*, *SketchBench* suffers from ambiguity recognition issues, such as overlapped and overtraced symbols, intrinsic in dense sketches.



(**a**)

Figure 15. *Cont.*



(**b**)

Figure 15. Distribution of answers to Questionnaire 1 by group: task *t1* (**a**) and task *t2* (**b**).

Similar ratings have been obtained for questions 1.2–1.5 regarding the edited class diagram symbols. In particular, for task *t1*, we observe that the Aggregation and Inheritance symbols are the symbols more difficult to sketch, since they are very similar and *SketchBench* confuses their recognition. On the other hand, for ArgoUML the Aggregation is the more difficult to draw since it is not included in the symbol palette, but it can be defined by drawing an Association symbol and by setting its properties through a context menu. For task *t2*, the difficulties in the sketching are increased for all symbols; this is mainly due to the arrangement of the symbols on the sheet. In particular, we observed that, while ArgoUML provides a support in the symbol layout, *SketchBench* users incrementally sketch the symbols on the sheet according to their vision of the solution and this makes the recognition process more difficult for complex sketches.

The answers to question 1.6 reveal that both tools produce predictable results. Finally, by analyzing the ratings for questions 1.7 and 1.8 we observe that when the task is not difficult, *SketchBench* is lesser difficult to use and easier to find the solution than ArgoUML. Indeed, for task *t2* the little practice of the students with sketching created frustration, which caused the drawing of diagrams difficult to recognize.

The boxplot in Figure 16 compares the distribution of student group answers to the conclusive questionnaire. The answers to question 2.1 highlight that students prefer to use *SketchBench* wrt. ArgoUML for both tasks. This consideration is slightly stronger for the students of group B probably because they used *SketchBench* on the easier task.

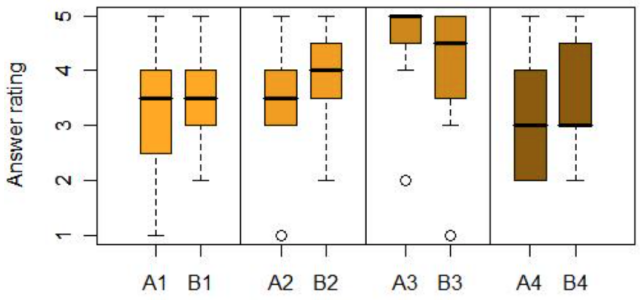


Figure 16. Distribution of answers to Questionnaire 2.

The ratings for question 2.2 reveal that *SketchBench* is easy to use for designing UML class diagrams. However, the complexity of the diagrams to be constructed influences this measure. In particular, we observe that the students had difficulties in fixing recognition mistakes when the diagrams contained many symbols. The outlier of group A corresponds to the student that was not able to create a recognizable diagram. The ratings for question 2.3 reveal that *SketchBench* is very

pleasant to use. The two outliers are the students frustrated by the inability of *SketchBench* to recognize their sketches. The ratings for question 2.4 reveal that the use of *SketchBench* requires the same mental effort of ArgoUML.

The main difficulties encountered by the students in the use of *SketchBench* and highlighted by the answers to question 2.5 include the way to report and fix errors in the sketches and the dual mode drawing approach used to separate drawings from text. The appealing features of *SketchBench* highlighted by answers to question 2.6 are the possibility of reasoning on the solution directly on the tool, the rapidity to create a diagram without selecting symbols from palette, and the satisfaction of recognition performances of *SketchBench*.

Overall, the answers to the questions reveal that both groups prefer *SketchBench* to ArgoUML for its easiness of use and pleasantness to use. Nevertheless, the results of this usability study should be further validated by considering other application domains and by comparing *SketchBench* with other sketch-based tools.

8. Conclusions and Future Work

We have presented a grammar-based approach for multi-domain hand-drawn diagram recognition. The recognition system is composed of three layers that perform context-based disambiguation and ink parsing. The domain symbols are recognized through their geometric properties allowing users the freedom to sketch and edit diagrams as they would naturally, while still making it possible to maintain a high level of recognition accuracy. The proposed recognition system performs a fine-to-coarse, incremental, and unobtrusive interpretation of the sketches as they are created.

The performance of the proposed approach has been evaluated in the domain of UML class diagrams. The obtained results clearly show the improvements with respect a baseline approach and demonstrate good scalability to larger drawings. We have also evaluated the usability of *SketchBench* to analyze the advantages of our tool wrt. traditional WIMP tools. The results have highlighted that *SketchBench* is preferred wrt. ArgoUML for easiness of use and pleasantness to use.

In general, a limit of the proposed approach is the process of grammar definition. Indeed, the grammar rules should be explicitly defined and compiled by the user. We intend to investigate inference procedures for obtaining a grammar from a set of sketch samples automatically in order to reduce this effort, as done in [63,64]. This also involves the definition of a technique for automatically setting the thresholds and the parameters included the proposed grammar formalism. We can observe that such parameters are similar to the probabilities associated to the productions of SCFG and many approaches for training stochastic grammars over a corpus have been proposed in the literature [65,66]. However, the parameters of each SCFG reflect the frequency of applying grammar rules in the recognition of training corpus. On the contrary, a discriminant value is associated to a terminal symbol and reflects the frequency of using such a terminal in the recognition of the various language symbols. In particular, the greater the frequency of using a terminal is, the smaller the discriminant value. Thus, the algorithms for estimating SCFGs should be revised to the different mean of such parameter in order to infer discriminant values from training corpus.

Another limitation of grammar-based approaches lies in the management of uncertainty expressed through over-sketching [67]. Indeed, it is difficult to encode this kind of inaccuracies in grammar productions. Noise and over-sketching are pervasive in hand-drawn diagrams of domains whose symbols have a complex geometry, such as petroglyph symbols [68,69]. To mitigate this issue, we plan to extend the proposed approach by integrating statistical information in the recognition process.

In the future, we also plan to further empirically validate the scalability of the approach by considering a more complex domain, such as military, engineering, and CAD symbology. Such domains are characterized by more complex symbols, a vocabulary size very great, and theyinclude various kinds of symbol relationships.

Author Contributions: Conceptualization, formal analysis, investigation, methodology, writing—original draft, writing—review & editing: V.D., Conceptualization, methodology, software, data curation, data analysis, writing—original draft: M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SkGs	Sketch Grammars
SCFG	stochastic context-free grammars
UML	Unified Modeling Language
HMM	Hidden Markov Model

Appendix A. The Parsing Program Algorithm

Input: A sequence of interpreted shapes, a parser tree, and a parsing table.
Output: A bottom-up analysis of the input sketch if it is syntactically correct, an error message otherwise.
Method: Start with the state s_0 on the top of the stack.

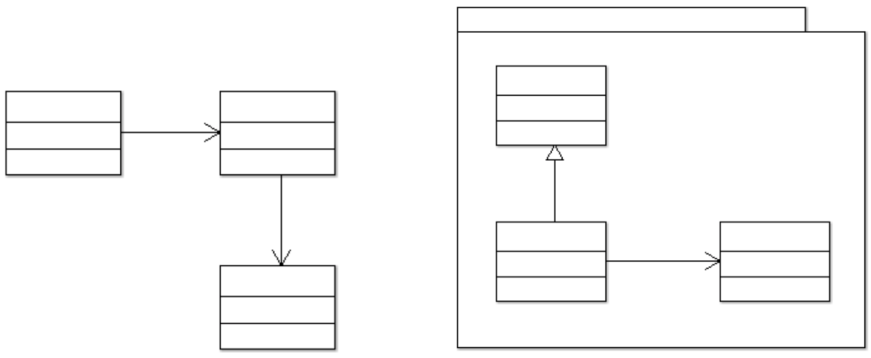
```
repeat
{
    s = stack[parser].currentState;
    if (fetch(next[s]) != null) { }
      else if (∃ a substate sb ∈ s, ∃' fetch(next[sb]) != null) { s = first_valid_substate(s); }
      else emit "syntax error" and exit;

    ip = fetch(next[s]);
    b = ip.grammarSymbol;

    if (action[s, b] == "accept") emit "success" and exit;
    else if (action [s, b] is a nonempty sequence seq of conditioned shifts with reduce of type "R_t : shift reduce x")
    {
        while (!seq. empty())
        {
            "R_t : shift reduce x" = pop_ first_element(seq);
            if (R_t.empty() || ( Test( REL^h, b )==true, foreach(REL^h ∈ R_t)))
            {
                stack[parser].push(b);
                stack[parser].push(-1);
                applyReduce(x);
            }
        }
    }
    else if (action [s, b] is a nonempty sequence seq of conditioned shifts of type "R_t : shift s'")
    {
        while (!seq. empty())
        {
            "R_t : shift s' " = pop_ first_element(seq);
            if (R_t.empty() || ( Test( REL^h, b )==true, foreach(REL^h ∈ R_t)))
            {
                stack[parser].push(b);
                stack[parser].push(s');
            }
        }
    }
    else if (action [s, b] is "reduce x")
    {
        applyReduce(x);
    }
     else emit "syntax error" and exit;

   if (no element has been pushed on or popped from the stack) emit "syntax error" and exit;
}
```

```
applyReduce(x)
{
  if (x == "(A_Γ → x1(p1) R1 x2(p2) R2 … x_{m-1}(p_{m-1}) R_{m-1} x_m(p_m), Action)" )
   {
     compute(Action);
     if(!Γ.empty()) compute(Γ);
     stack[parser].pop(2*m);
     s' = stack[parser].currentState;
     applyGoto(s', A);
   }
}
```

```
applyGoto(s', A)
{
   if (goto[s', A] is a nonempty sequence seq of conditioned gotos of type "Rt : s")
   {
       while (!seq.empty())
       {
          "Rt : s'" = pop_ first_element(seq);
          if (Rt.empty() || ( Test( RELh, A )==true, foreach(RELh ∈ Rt)))
          {
                  stack[parser].push(A);
                  stack[parser].push(s");
          }
          }
    }
   else if (goto[s', A] is a nonempty sequence seq of conditioned gotos of type "Rt : reduce x")
   {
       while (!seq.empty())
       {
          "Rt : reduce x" = pop_ first_element(seq);
          if (Rt.empty() || ( Test( RELh, A )==true, foreach(RELh ∈ Rt)))
          {
                  stack[parser].push(A);
                  stack[parser].push(-1);
                  applyReduce(x);
          }
       }
    }
}
```

Appendix B

In the following we report the five class diagrams given to the users for the evaluation of the recognition system described in Section 7.



Class Diagram D1 (**a**)

Class Diagram D2 (**b**)

Figure A1. *Cont.*

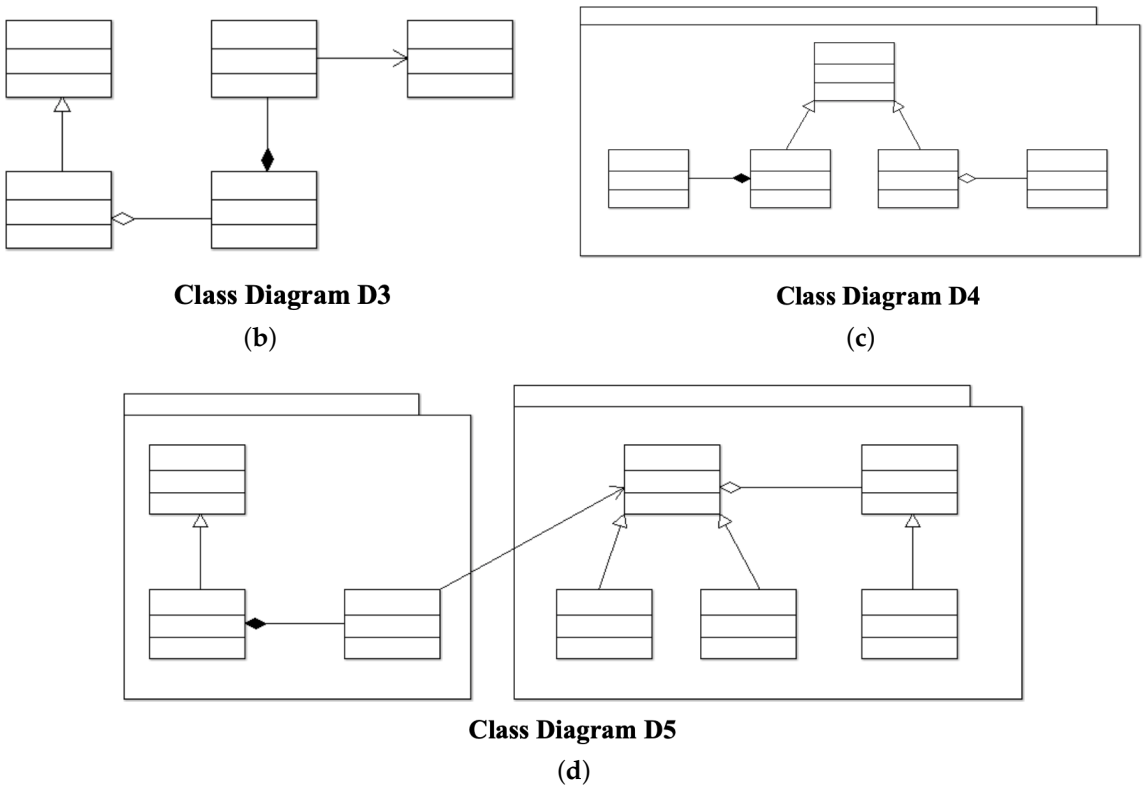


Figure A1. The five class diagrams to be drawn in an unconstrained fashion by the twenty subjects using *SketchBench*.

References

1. Davis, R. Magic Paper: Sketch-Understanding Research. *IEEE Comput.* **2007**, *40*, 34–41.
2. Igarashi, T.; Zeleznik, B. Sketch-based Interaction. *IEEE Comput. Graph. Appl.* **2007**, *27*, 26–27.
3. Stahovich, T.; Davis, R.; Shrobe, H. Generating Multiple New Designs from a Sketch. *Artif. Intell.* **1998**, *104*, 211–264.
4. Ghorbel, A.; Lemaitre, A.; Anquetil, E.; Fleury, S.; Jamet, E. Interactive interpretation of structured documents: Application to the recognition of handwritten architectural plans. *Pattern Recognit.* **2015**, *48*, 2446–2458.
5. Atilola, O.; Valentine, S.; Kim, H.H.; Turner, D.; McTigue, E.; Hammond, T.; Linsey, J. Mechanix: A natural sketch interface tool for teaching truss analysis and free-body diagrams. *Artif. Intell. Eng. Des. Anal. Manuf.* **2014**, *28*, 169–192.
6. Goel, V. *Sketches of Thought*; The MIT Press: Cambridge, MA, USA, 1995.
7. Hearst, M. Sketching Intelligent Systems. *IEEE Intell. Syst. Their Appl.* **1998**, *13*, 10–19.
8. Igarashi, T.; Matsuoka, S.; Kawachiya, S.; Tanaka, H. Interactive Beautification: A Technique for Rapid Geometric Design. In *Proceedings of UIST'97*; ACM Press: New York, NY, USA, 1997; pp. 105–114.
9. Cheema, S.; LaViola, J.J. Applying Mathematical Sketching to Sketch-Based Physics Tutoring Software. In *Smart Graphics*; Taylor, R., Boulanger, P., Krüger, A., Olivier, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 13–24.
10. Fišer, J.; Asente, P.; Schiller, S.; Sýkora, D. Advanced drawing beautification with ShipShape. *Comput. Graph.* **2016**, *56*, 46–58.
11. Rubine, D. Specifying Gestures by Example. *Comput. Graph.* **1991**, *25*, 329–337.
12. Damm, C.; Hansen, K.; Thomsen, M. Tool Support for Cooperative Object-oriented Design: Gesture based Modeling on an Electronic Whiteboard. *Chi Lett.* **2000**, *2*, 518–525.
13. Li, L.; Fu, H.; Tai, C. Fast sketch segmentation and labeling with deep learning. *IEEE Comput. Graph. Appl.* **2019**, *39*, 38–51.

14. Schneider, R.G.; Tuytelaars, T. Example-Based Sketch Segmentation and Labeling Using CRFs. *ACM Trans. Graph.* **2016**, *35*, doi:10.1145/2898351.
15. Lank, E.; Thorley, J.; Chen, S. An Interactive System for Recognizing Hand Drawn UML Diagrams. In Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'00), Mississauga, ON, Canada, 13–16 November 2000; pp. 1–15.
16. Sezgin, T. Online Sketch Recognition from a Dynamic Perspective. Ph.D. Dissertation, Department Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.
17. Wenyin, L.; Qian, W.; Xiao, R.; Jin, X. Smart Sketchpad–An On-line Graphics Recognition System. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'01), Seattle, WA, USA, 13 September 2001; IEEE CS Press: Los Alamitos, CA, USA, 2001; pp. 1050–1054.
18. Costagliola, G.; Deufemia, V.; Risi, M. Sketch Grammars: A Formalism for Describing and Recognizing Diagrammatic Sketch Languages. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'05), Seoul, Korea, 31 August–1 September 2005; IEEE CS Press: Los Alamitos, CA, USA, 2005.
19. Costagliola, G.; Deufemia, V.; Risi, M. A Multi-layer Parsing Strategy for On-line Recognition of Hand-drawn Diagrams. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06), Brighton, UK, 4–8 September 2006; IEEE CS Press: Los Alamitos, CA, USA, 2006; pp. 103–110.
20. Aho, A.; Sethi, R.; Ullman, J. *Compilers Principles, Techniques, and Tools*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1987.
21. Snelting, G. How to Build LR Parsers which Accept Incomplete Input. *Sigplan Not.* **1990**, *25*, 51–58.
22. Kara, L.; Stahovich, T. Hierarchical Parsing and Recognition of Hand-sketched Diagrams. In *Proceedings of UIST'04*; ACM Press: New York, NY, USA, 2004; pp. 13–22.
23. McFadzean, J.; Cross, N.G.; Johnson, J.H. An Analysis of Architectural Visual Reasoning in Conceptual Sketching Via Computational Sketch Analysis (CSA). In Proceedings of the IEEE International Conference on Information Visualization, London, UK, 14–16 July 1999; pp. 258–265.
24. Apte, A.; Vo, V.; Kimura, T.D. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In *Proceedings of UIST'93*; ACM Press: New York, NY, USA, 1993; pp. 121–128.
25. Fonseca, M.; Pimentel, C.; Jorge, J. CALI - An Online Scribble Recognizer for Calligraphic Interfaces. In *AAAI Spring Symposium on Sketch Understanding*; AAAI Press: Cambridge, MA, USA, 2002; pp. 51–58.
26. Sun, Z.; Jiang, E.; J., S. Adaptive Online Multi-Stroke Sketch Recognition based on Hidden Markov Model. *Adv. Mach. Learn. Cybern.* **2006**, *3930*, 948–957.
27. Kara, L.; Stahovich, T. An Image-based, Trainable Symbol Recognizer for Hand-drawn Sketches. *Comput. Graph.* **2005**, *29*, 501–517.
28. Hu, R.; James, S.; Wang, T.; Collomosse, J. Markov random fields for sketch based video retrieval. In *Proceedings of ACM Conference on International Conference on Multimedia Retrieval*; ACM Press: New York, NY, USA, 2013; pp. 279–286.
29. Schneider, R.G.; Tuytelaars, T. Sketch Classification and Classification-Driven Analysis Using Fisher Vectors. *ACM Trans. Graph.* **2014**, *33*, 1–9.
30. Deufemia, V.; Risi, M.; Tortora, G. Sketched symbol recognition using Latent-Dynamic Conditional Random Fields and distance-based clustering. *Pattern Recognit.* **2014**, *47*, 1159–1171.
31. Seddati, O.; Dupont, S.; Mahmoudi, S. DeepSketch: Deep convolutional neural networks for sketch recognition and similarity search. In Proceedings of the 2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI), Prague, Czech Republic, 10–12 June 2015; IEEE CS Press: Los Alamitos, CA, USA, 2015; pp. 1–6.
32. Zhang, H.; Liu, S.; Zhang, C.; Ren, W.; Wang, R.; Cao, X. SketchNet: Sketch classification with web images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*; IEEE CS Press: Los Alamitos, CA, USA, 2016; pp. 1105–1113.
33. Zhang, X.; Huang, Y.; Zou, Q.; Pei, Y.; Zhang, R.; Wang, S. A Hybrid convolutional neural network for sketch recognition. *Pattern Recognit. Lett.* **2020**, *130*, 73–82.
34. Saund, E.; J., M.; Fleet, D.; Larner, D.; Lank, E. Perceptual Organization as a Foundation for intelligent Sketch Editing. In *Proceedings of AAAI Spring Symposium on Sketch Understanding*; AAAI Press: Cambridge, MA, USA, 2002; pp. 118–125.

35. Lank, E.; Saund, E. Sloppy Selection: Providing an Accurate Interpretation of Imprecise Stylus Selection Gestures. *Comput. Graph.* **2005**, *29*, 490–500.
36. Perteneder, F.; Bresler, M.; Grossauer, E.M.; Leong, J.; Haller, M. CLuster: Smart Clustering of Free-Hand Sketches on Large Interactive Surfaces. In Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, ACM, UIST'15, Charlotte, NC, USA, 8–11 November 2015; pp. 37–46.
37. Sezgin, T.; Davis, R. Sketch Interpretation Using Multiscale Models of Temporal Patterns. *IEEE Comput. Graph. Appl.* **2007**, *27*, 28–37.
38. Hall, A.; Pomm, C.; Widmayer, P. A Combinatorial Approach to Multi-Domain Sketch Recognition. In Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM'07), Riverside, CA, USA, 2–4 August 2007.
39. Bresler, M.; Průša, D.; Hlaváč. Online recognition of sketched arrow-connected diagrams. *Int. J. Doc. Anal. Recognit.* **2016**, *19*, 253–267.
40. Alvarado, C.; Davis, R. Dynamically Constructed Bayes Nets for Multi-Domain Sketch Understanding. In Proceedings of the IJCAI'05, Edinburgh, UK, 30 July–5 August 2005; pp. 1407–1412.
41. Hammond, T.; Davis, R. LADDER—A Sketching Language for User Interface Developers. *Comput. Graph.* **2005**, *29*, 518–532.
42. Stiny, G.; Gips, J. Shape Grammars and the Generative Specification of Painting and Sculpture. *Inf. Process.* **1972**, *71*, 1460–1465.
43. Mahoney, J.; Fromherz, M. Three Main Concerns in Sketch Recognition and an Approach to Addressing Them. In *Proceedings of AAAI Spring Symposium on Sketch Understanding*; AAAI Press: Cambridge, MA, USA, 2002; pp. 105–112.
44. Pasternak, B. Processing Imprecise and Structural Distorted Line Drawings by an Adaptable Drawing Interpretation Kernel. In Proceedings of the IAPR Workshop on Document Analysis Systems (DAS'94), Kaiserslautern, Germany, 18–20 October 1994; pp. 349–363.
45. Landay, J. Interactive Sketching for the Early Stages of User Interface Design. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1996.
46. Shizuki, B.; Yamada, H.; Iizuka, K.; Tanala, J. A Unified Approach for Interpreting Handwritten Strokes. In Proceedings of the IEEE Symposium on Human-Centric Computing, Auckland, New Zealand, 31 October 2003; IEEE CS Press: Los Alamitos, CA, USA, 2003; pp. 180–182.
47. Shilman, M.; Pasula, H.; Russell, S.; Newton, R. Statistical Visual Language Models for Ink Parsing. In *Proceedings of AAAI Spring Symposium on Sketch Understanding*; AAAI Press: Cambridge, MA, USA, 2002; pp. 126–132.
48. Álvaro, F.; Sánchez, J.A.; Benedí, J.M. Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models. *Pattern Recognit. Lett.* **2014**, *35*, 58–67.
49. Macé, S.; Anquetil, E. Eager interpretation of on-line hand-drawn structured documents: The DALI methodology. *Pattern Recognit.* **2009**, *42*, 3202–3214.
50. Krichen, O.; Girard, N.; Anquetil, É.; Corbille, S.; Renault, M. Real-Time Interpretation of Hand-Drawn Sketches with Extended Hierarchical bi-Dimensional Grammar. In Proceedings of the 16th International Conference on Frontiers in Handwriting Recognition, IEEE Computer Society, ICFHR 2018, Niagara Falls, NY, USA, 5–8 August 2018; pp. 273–278.
51. Object Management Group. UML Specification Version 2.5.1. Technical Report. Available online: https://www.omg.org/spec/UML (accessed on 14 August 2020).
52. Duda, R.; Hart, P. *Pattern Classification and Scene Analysis*; Wiley Press: New York, NY, USA, 1973.
53. Hse, H.; Shilman, M.; Newton, A. Robust Sketched Symbol Fragmentation using Templates. In *Proceedings of IUI '04*; ACM Press: New York, NY, USA, 2004; pp. 156–160.
54. Zhang, X.; Song, J.; Dai, G.; Lyu, M. Extraction of Line Segments and Circular Arcs from Freehand Strokes based on Segmental Homogeneity Features. *IEEE Trans. Syst. Man Cybern.* **2006**, *36*, 300–311.
55. Yin, L.; Yajie, Y.; Weyin, L. Online Segmentation of Freehand Stroke by Dynamic Programming. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'05), Seoul, Korea, 31 August–1 September 2005; IEEE CS Press: Los Alamitos, CA, USA, 2005; pp. 197–201.
56. Weisstein, E. Least Squares Fitting–Perpendicular Offsets. Technical Report. Available online: http://mathworld.wolfram.com/LeastSquaresFittingPerpendicularOffsets.html (accessed on 14 August 2020).

57. Fitzgibbon, A.; Pilu, M.; Fisher, R. Direct Least Square Fitting of Ellipses. *IEEE Trans. Pattern Anal. Mach. Intell.* **1999**, *21*, 476–480.
58. Costagliola, G.; Deufemia, V.; Polese, G.; Risi, M. A Parsing Technique for Sketch Recognition Systems. In Proceedings of the IEEE Symposium VL/HCC'04, Rome, Italy, 26–29 September 2004; pp. 19–26.
59. Costagliola, G.; Deufemia, V.; Polese, G.; Risi, M. Building Syntax-Aware Editors for Visual Languages. *J. Vis. Lang. Comput.* **2005**, *16*, 508–540.
60. Costagliola, G.; Deufemia, V.; Polese, G. A Framework for Modeling and Implementing Visual Notations with Applications to Software Engineering. *Acm Trans. Softw. Eng. Methodol.* **2004**, *13*, 431–487.
61. Costagliola, G.; Deufemia, V.; Risi, M. Using Error Recovery Techniques to Improve Sketch Recognition Accuracy. In Proceedings of the 7th International Workshop on Graphics Recognition, GREC'07, Curitiba, Brazil, 20–21 September 2007; pp. 157–168.
62. Salton, G.; McGill, M. *Introduction to Modern Information Retrieval*; McGraw-Hill: New York, NY, USA, 1983.
63. Romeu, J.; Lamiroy, B.; Sanchez, G.; Llados, J. Automatic Adjacency Grammar Generation from User Drawn Sketches. In Proceedings of the International Conference on Pattern Recognition (ICPR'06), Hong Kong, China, 20–24 August 2006; IEEE CS Press: Los Alamitos, CA, USA, 2006; pp. 1026–1029.
64. Veselova, O.; Davis, R. Perceptually based Learning of Shape Descriptions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*; AAAI Press: Cambridge, MA, USA, 2004; pp. 482–487.
65. Benedì, J.; Sànchez, J. Estimation of stochastic context-free grammars and their use as language models. *Comput. Speech Lang.* **2005**, *19*, 249–274.
66. Ra, D.Y.; Stockman, G. A New one Pass Algorithm for Estimating Stochastic Context-Free Grammars. *Inf. Process. Lett.* **1999**, *72*, 37–45.
67. Carton, C.; Lemaitre, A.; Coüasnon, B. Fusion of Statistical and Structural Information for Flowchart Recognition. In Proceedings of the 12th International Conference on Document Analysis and Recognition, IEEE Computer Society, 2013, ICDAR'13, Washington, DC, USA, 25–28 August 2013; pp. 1210–1214.
68. Deufemia, V.; Paolino, L.; de Lumley, H. Petroglyph Recognition Using Self-Organizing Maps and Fuzzy Visual Language Parsing. In Proceedings of the IEEE 24th International Conference on Tools with Artificial Intelligence, IEEE Computer Society, ICTAI'12, Athens, Greece, 7–9 November 2012; pp. 852–859.
69. Deufemia, V.; Paolino, L.; Tortora, G.; Traverso, A.; Mascardi, V.; Ancona, M.; Martelli, M.; Bianchi, N.; de Lumley, H. Investigative analysis across documents and drawings: Visual analytics for archaeologists. In Proceedings of the International Working Conference on Advanced Visual Interfaces, ACM, AVI'18, Naples, Italy, 22–25 May 2012; pp. 539–546.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).