

SketchML a Representation Language for Novel Sketch Recognition Approach

Danilo Avola
Univ. of Rome "La Sapienza"
Dept. of Computer Science
Via Salaria 113, 00189
Rome, Italy
avola@di.uniroma1.it

Andrea Del Buono
National Research Center
CSKLab Research Dept.
Via Savoia 84, 00196
Rome, Italy
delbuono@csklab.it

Giorgio Gianforme
Univ. of Rome "Rome 3"
Dept. of Computer Science
Via Vasca Navale 79, 00146
Rome, Italy
gianforme@dia.uniroma3.it

Stefano Paolozzi
Univ. of Rome "Rome 3"
Dept. of Computer Science
Via Vasca Navale 79, 00146
Rome, Italy
paolozzi@dia.uniroma3.it

Rui Wang
Univ. of Germany "Saarland"
Dept. of Computer Science
Saarbrücken, 66041
Saarbrücken, Germany
rwang@coli.uni-sb.de

ABSTRACT

Multimodal interfaces can be profitably used to support increasingly complex services in assistive environments. In particular, sketch-based interfaces offer users an effortless and powerful communication way to represent concepts and commands on different devices. Unlike other modalities, sketch-based interaction can be easily fitted according to heterogeneous services. Moreover it can be quickly personalized according to the user needs.

Developing a sketch-based interface for a specific service is a time-consuming operation that requires the re-engineering and/or the re-designing of the whole recognizer framework. This paper describes a definitive framework by which the user, simply by using freehand drawing, can define every kind of sketch-based interface. The definition of the interface and its recognition process are performed by using our developed Sketch Modeling Language (*SketchML*).

Categories and Subject Descriptors

H.5 [Information Interfaces and Presentation]: Miscellaneous; D.2.8 [Pattern Recognition]: Design Methodology—*feature evaluation and selection, pattern analysis.*

General Terms

Algorithms, experimentation.

Keywords

Sketch-based interfaces, sketch-based interaction, sketch recognition, multi-domain, vectorization, segmentation, XML, SVG.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PETRA '09, June 09-13, 2009, Corfu, Greece.

Copyright 2009 ACM ISBN 978-1-60558-409-6 ...\$5.00.

1. INTRODUCTION

Multimodal interfaces allow users to interact naturally with any desktop or mobile device using multiple modalities (e.g. sketch, gesture, speech, gaze). In assistive environments these interfaces can be profitably used to support increasingly complex services able to improve the quality of life. In particular, sketch-based interfaces enable users to express concepts, provide commands, and represent ideas in an immediate, intuitive and simple way. In fact, the users can access any kind of device and related services by drawing a set of 2D graphical symbols (e.g. arrows, geometric symbols, or more complex shapes). In this context a set of 2D graphical symbols is also called library or application domain. Unlike other modalities, sketch-based interfaces allow users easier customization and quick personalization of the interaction. The first aspect highlights the simplicity to draw well known standard graphical symbols (e.g. commands) to access specific services. The second aspect focuses on the suitability to use drawn symbols (e.g. commands and/or concepts) to access services according to user needs and/or capacities.

Developing a sketch-based interface able to distinguish each symbol that makes up a library is a time-consuming operation that requires the re-engineering and/or the re-designing of the whole recognizer framework. This paper describes a definitive framework by which the user, simply by using freehand drawing, can define every kind of sketch-based interface based on 2D graphical symbols. The definition of the interface and its recognition process are performed by using our developed Sketch Modeling Language (*SketchML*).

The developed framework, compared to the current prototypes, has three advantageous main features that jointly put it in the vanguard in the sketch-based interfaces area. The first regards the ability to define every kind of library. In fact, several frameworks are designed to properly recognize only a specific set of symbols: they could also recognize other kinds of libraries but this step would involve the arrangement of the recognition engine inside the frameworks. The second feature regards the library definition, simply by using freehand drawing. In fact, there are several

frameworks based on multi-domain concept, but to build a new application domain it is necessary an expert user with a deep knowledge about standards, vectorial applications, programming languages and so on. Using our framework the user, without special knowledge, can create any kind of 2D graphical symbols. The last feature concerns the definition of the interface and its recognition process which are performed by using SketchML, a language based on XML (eXtended Markup Language) standard. This consolidated W3C (World Wide Web Consortium) standard allows both compatibility on different devices and cooperation between different applications/services.

The paper is structured as follows. Section 2 proposes some remarkable related works in multi-domain sketch-based interfaces area. Section 3 introduces an overview about the developed framework. Section 4 shows the library editor environment. Section 5 describes the sketch recognition environment. Section 6 discusses about the matching between the library symbols and the user-performed sketch. Section 7 shows experimental results on a wide application domain designed to meet the needs of assistive environments. Finally, Section 8 concludes the paper.

2. RELATED WORKS

This section shows some remarkable works about the multi-domain sketch recognition frameworks. It is important to observe, once again, that our further contribution, on this area, is to provide a framework able to create every kind of library, simply by using the freehand drawing activity. In this way, without theoretical and technical specific knowledge a user can build, in real time, every kind of simple or complex library. Moreover, all the libraries are represented using the developed XML-based language (SketchML) which allows both compatibility on different devices and cooperation between different applications/services. In this context, it has to be highlighted that also dedicated devices, in assisted living, tend to integrate XML language interpreters in order to allow exchange of information and functionalities between different devices and applications and services. An interesting framework about multi-domain sketch recognition is presented in [3]. In this work the authors show a free-sketch recognition framework that allows users to draw shapes in natural way. In particular, the framework is able to recognize a variety of defaults domains by exploiting several well known techniques able to catch information about: geometric, features, temporal, contextual, multi-modal and domain. A different approach is provided in [1]. In this work the authors present a novel form of dynamically constructed Bayes net, developed for multi-domain sketch recognition. The proposed sketch recognition engine integrates shape information and domain knowledge to improve recognition process. In [4] the authors show a remarkable agent-based framework for sketched symbol interpretation that heavily exploits contextual information for ambiguity resolution. In particular, in this work the agents manage the activity of low-level hand-drawn symbol recognizers and coordinate themselves in order to exchange contextual information, thus leading to an efficient and precise interpretation of sketches. Another interesting feature-based approach is shown in [7]. In this work the authors present an advanced heuristic based framework that offers the user more liberty for free-style sketching as well as for grouping the strokes, reducing the complexity for recognizing the sketches. Un-

like previous approaches the authors in [10] introduce a primitive-based engine. In particular, the authors present a method for interpreting on-line freehand sketch and describe a human-computer interface prototype system that is designed to infer designer's intention and interprets the input sketch into more exact geometric primitives: straight lines, poly-lines, circles, circular arcs, ellipses, elliptical arcs, and parabolas. An important work, tied to our approach, is explained in [5]. The authors introduce LADDER, the first language to describe how sketched diagrams in a domain are drawn, displayed, and edited. Also in this work the difficulties about a suitable choice of a set of predefined entities that is broad enough to support a wide range of domains are faced. This language consists of predefined shapes, constraints, editing behaviors, and display methods, as well as a syntax for specifying a domain description sketch grammar and extending the language, ensuring that shapes and shape groups from many domains can be described. The just mentioned work has been a real source of inspiration. Our proposed SketchML has several common points with LADDER language, but SketchML is more general and it can be used to represent any application domain. Moreover, our approach allows users to build any kind of library, simply by using freehand drawing activity. A particular example is shown in [11], in this work the authors describe a domain-independent system for sketch recognition. The developed system allows users to draw sketches as naturally as they do on paper, and it recognizes the drawing through imprecise stroke approximation which is implemented in a unified and incremental procedure. A different approach is shown in [2], which describes a framework for recognizing of composite graphic objects, highlighting the important role played by topological spatial relationships between their components. In this work the authors introduce a ternary relationship, which is a complement to a binary relationship, to describe composite graphic objects. Another remarkable approach is shown in [9]. In this work the authors describe a statistical framework based on dynamic Bayesian networks that explicitly models the fact that objects can be drawn interspersed. We conclude this section showing two approaches based on HMMs (Hidden Markov Models). In fact, a new interesting tendency on sketch recognition approaches is to use HMM to overcome several critical duties, such as: personalization, graphical noise, shape learning, and so on. For example, a simple and innovative approach is proposed in [8]. In this work the authors show that it is possible to view sketching as an interactive process in order to use HMM for sketch recognition modeling. A different approach is proposed in [6], in this interesting work the authors describe a framework for on-line multi-stroke sketch recognition. By using the framework, user sketches are modeled to HMM chains, and strokes are mapped to different HMM states. The authors have introduced a new method to determine HMM state-number, based on which an adaptive HMM sketch recognizer is constructed.

The mentioned approaches depend on a set of default symbols and/or primitives that are tied to specific application domains, or they are tied to certain statistical models. Moreover, in both cases, the building of a library is a hard task that has to be performed by skilled user. Our novel approach overcomes these problems by using very general primitives. Moreover every kind of library can be built simply by using freehand sketch activity.

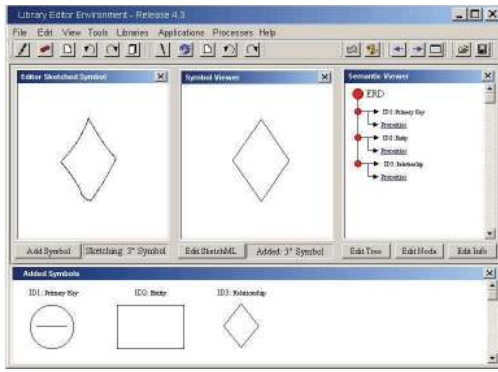


Figure 1: Library Editor Environment.

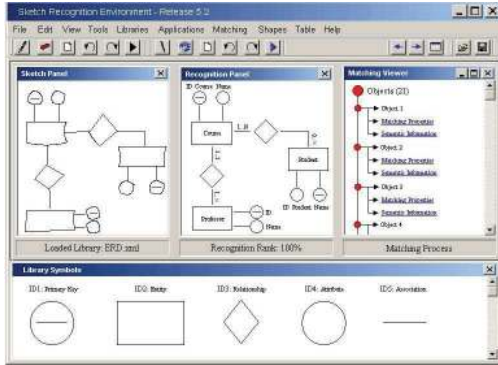


Figure 2: Sketch Recognition Environment.

3. DEVELOPED FRAMEWORK

The developed framework provides two different environments. The first is the Library Editor Environment (LEE), where the user can define the libraries by means of freehand drawing. The second is the Sketch Recognition Environment (SRE), where the system performs the recognition activity, on a user-drawn sketch, after loading the related library. In order to explain the functionalities of the two just mentioned environments, in this section we will give a simple example based on ERD (Entity Relationship Diagram).

As shown in Figure 1 the user has sketched the third symbol belonging to the ERD domain, and the LEE has built the related vectorial representation. The process (shown in section 4) for obtaining the vectorial symbol from the sketched symbol is the main aspect of our paper. In Figure 2 a recognition process (shown in section 5) about a simple ERD example is given. In order to understand the matching process (shown in section 6) it is important to observe that both vectorial representation of the library symbols built in LEE and vectorial diagram obtained by recognition process in SRE are expressed by our XML-based developed language: SketchML. The use of sketch-based interfaces allow the users, both in LEE and SRE, to perform several useful activities, such as: deletions, restyling of the whole symbol/object (set of symbols) or part of it, and so on.

SRE exploits the knowledge of the symbols that make up the loaded library to improve the recognition process. In fact, as just mentioned, the final result of the library definition is an XML file that describes (by SketchML) each and every symbol. Also the final result of the recognized user sketch

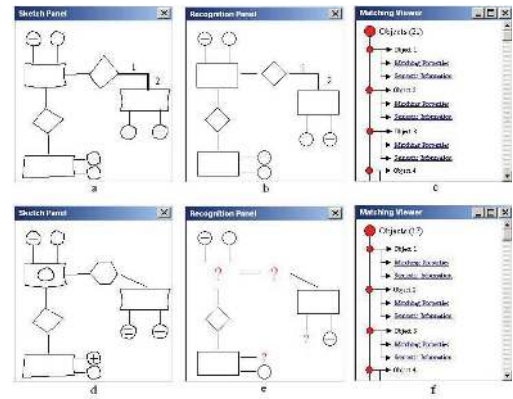


Figure 3: Correct/Incorrect ERD Example.

is an XML file that describes the whole sketch performed by the user. But this last file takes into account only the recognized symbols according to the loaded library. In order to explain the just mentioned concepts it is important to introduce the example in Figure 3, that is based on the five symbols belonging to the ERD library.

In Figure 3-a the user has correctly sketched an ERD example. More exactly, the user has sketched the following sequence of ERD symbols: three primary keys, three entities, two relationships, three attributes and ten associations (twenty one total objects). In Figure 3-b, all the drawn objects have been recognized and an XML file (described in the next section by SketchML) that describes each one of the twenty one objects has been created. Indeed, as explained in the next sections too, the situation is more complex than one just mentioned. In fact, taking into account the symbols belonging to the loaded library, it is possible to observe that the sketched symbol of association (shown in Figure 3-a by labels 1 and 2) is not a library symbol. It is recognized as an association symbol because it results from the compositions of two symbols of the ERD library (that is two association symbols) with a particular constraint (an angle of ninety degrees). In Figure 3-c is shown the matching process (explained in the relative section) of each symbol. This process allows the framework to map every object sketched by the user (in SRE) with a symbol belonging to the related library (defined in LEE). Unlike the previous example, in Figure 3-d the user has incorrectly sketched an ERD example. In fact, as shown in Figure 3-e, seventeen objects have been recognized and four objects have not been recognized. In this case the XML file will contain only the description of the seventeen objects. In Figure 3-f is detailed each one of the matching process between the objects drawing by the user and the loaded library. This example highlights that four objects have not satisfied the matching requirements (in fact the matched objects are seventeen), this means that these four objects were not matched with a library symbol by the matching process, and there is no SketchML description for these objects.

In the next two sections show the approaches and the algorithms behind the two just introduced environments, in a further section the approach used to perform the matching process between a loaded library and the sketch performed by the user is given.

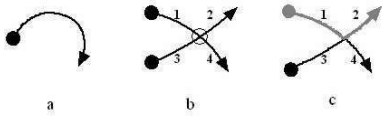


Figure 4: Stroke, Sub-Strokes, Generalized Strokes.

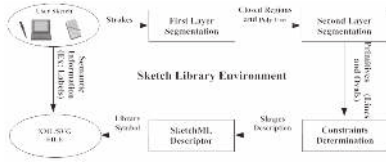


Figure 5: Library Editor Environment Schema.

4. LIBRARY EDITOR ENVIRONMENT

This section details approaches and algorithms that form the core of the LEE. In order to explain the section content, the following three definition are introduced.

- Stroke: set of pixels obtained during the following pen action: pen down, pen drawing, pen up.
- Sub-Stroke: subpart of a Stroke. When two (or more) strokes are crossed several intersection points are created. By these points a lot of sub-strokes can be identified.
- Generalized Stroke: it is made up from two (or more) sub-strokes coming from different strokes.

In Figure 4-a an example of stroke is shown. In particular, the circle indicates the start point of the stroke (pen down), and the arrowhead indicates the final point of the stroke (pen up). In Figure 4-b, by two start points, two end points, and the intersection point (shown by the empty circle) four sub-strokes are identified (respectively labeled by: 1, 2, 3 and 4). Finally, in Figure 4-c two generalized strokes are shown, more exactly the generalized stroke made up by sub-strokes labeled with 1 and 2, and that one made up by sub-strokes labeled with 3 and 4 (other combinations are available).

In Figure 5 a simple schema about the basic concepts that drive the making of a general library is given. The first module (First Layer Segmentation Algorithm) is used to obtain, from the strokes performed by the user during sketch activity, two kind of objects: closed regions and poly-lines. In the second module (Second Layer Segmentation Algorithm) each object is analyzed in order to detect the set of primitives that makes it up: ovals and lines. In the third module (Constraints Determination) the spatial relationships between the discovered primitives are evaluated. The role of the fourth module (SketchML Descriptor) is “to translate” all the found information in an XML/SVG structure.

Next subsections show each module of the proposed schema. Besides, the example in Figure 6 will attend each section in order to explain the involved processes. In particular, Figure 6-a shows a sketched symbol that the user wants to add to the library. Figure 6-b shows the related vectorial representation added to the library and described inside the framework by SketchML. Figure 6-c will be explained in the next subsection.

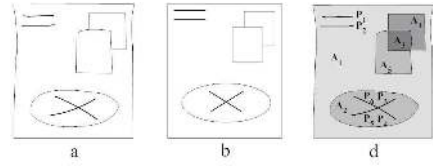


Figure 6: A Library Symbol Example.

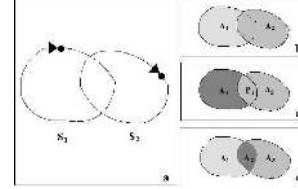


Figure 7: Ambiguity Examples.

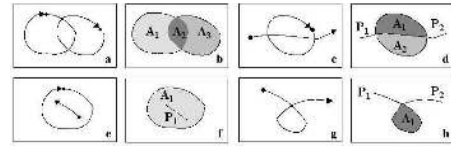


Figure 8: Algorithm Interpretation.

4.1 First Layer Segmentation

The aim of the first layer segmentation algorithm is to provide two different set of objects: closed regions and poly-lines. This concept is not so simple as it might appear. In fact, there is not a unique interpretation about the two sketched strokes (S_1 and S_2) drawn in Figure 7-a. For example, the two strokes could be interpreted as two overlapped closed regions (A_1 and A_2 , Figure 7-b), otherwise they could be interpreted as two closed regions and a poly-line (A_1 , A_2 and P_1 , Figure 7-c). In another interpretation the two strokes could be interpreted as three closed regions (A_1 , A_2 and A_3 , Figure 7-d). Moreover, there are other available interpretations. This kind of issues, defined as ambiguity problems, are a crucial point of every sketch recognition system. In order to overcome them, the first layer segmentation algorithm works taking into account the following two characterizations of closed region and poly-line.

- Closed Region: it is the smallest area confined by a set of strokes and/or sub-strokes and/or generalized strokes.
- Poly-Line: it is the smallest stroke or sub-stroke.

With these characterizations, every sketch can be interpreted in a unique way. To highlight the algorithm approach in Figure 8 some examples of developed algorithm interpretation are given. More specifically, the two strokes drawn in Figure 8-a are interpreted, as shown in Figure 8-b, as three closed regions (A_1 , A_2 , A_3). The two strokes drawn in Figure 8-c are interpreted, as shown in Figure 8-d, as two closed regions and two poly-lines (A_1 , A_2 , P_1 , P_2). The two strokes drawn in Figure 8-e are interpreted, as shown in Figure 8-f, as a closed region and a poly-line (A_1 , P_1). Finally, the stroke drawn in Figure 8-g is interpreted, as shown in Figure 8-h, as a closed region and two poly-lines (A_1 , P_1 , P_2).

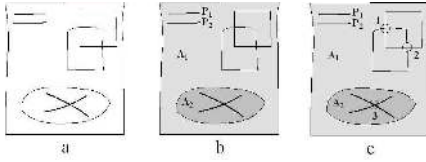


Figure 9: Closed Regions and Poly-Lines.

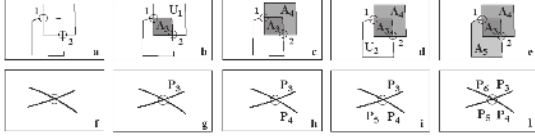


Figure 10: Exhaustive Sub-Algorithm.

On the example previously introduced to explain the whole LEE process (Figure 6) the just mentioned characterizations provide the following set of objects: five closed regions and six poly-lines (Figure 6-c).

More formally our algorithm works according to the following main process:

ALGORITHM 4.1.1. *The main step is to detect all the intersection points belonging to the sketch performed by the user. Subsequently, the following two sub-steps have to be performed:*

- (i) *every stroke, without intersection points, is analyzed from a simple sub-algorithm able to check if it is closed or not. The sub-algorithm performs this easy task looking for a relationship of proximity between start and end points of the stroke.*
- (ii) *every intersection point is used to look for the maximum number of disjointed areas. To accomplish this task an exhaustive sub-algorithm is performed. From every interaction point each possible closed path is identified. During this process the exhaustive sub-algorithm considers, step by step, only the smallest areas that not overlap (entirely or in part) other detected areas. In particular, the sub-algorithm will consider only the smallest closed path (i.e. the smallest length in pixels of the path that start and end on a same intersection point).*

In order to explain Algorithm 4.1.1, in Figure 9 a brief explication, about the involved steps, is given. By first point of the proposed algorithm (i) applied to the sketched symbol in Figure 9-a two closed regions and two poly-lines are found (as shown in Figure 9-b, A_1 , A_2 and P_1 , P_2). By the second step (ii), three intersection points are detected (as shown in Figure 9-c, 1, 2 and 3). In Figure 10 is shown how the exhaustive sub-algorithm, working on the intersection points, is able to detect the remaining closed regions and poly-lines. More specifically, in Figure 10-a the two intersection points about the crossed strokes are shown (1 and 2). As it is possible to observe in Figure 10-b the first intersection point (1) is used to detect the smallest path (i.e. the area A_3) that start and end on 1. After this choice the sub-algorithm takes into account two areas: the U_1 area and the union between the U_1 and A_3 areas. This second area is discarded because it covers (in part) a just found area (A_3), for this reason

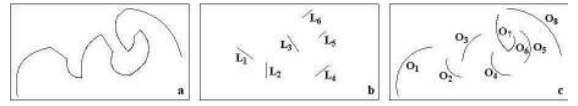


Figure 11: Primitives of a drawn stroke.

only the U_1 area, that once definitively detected is called A_4 , is considered (as shown in Figure 10-c). A very similar reasoning can be applied to the involved U_2 and A_3 areas (shown in Figure 10-d). In this way the maximum number of areas (A_3 , A_4 , A_5) can be detected by the sub-algorithm (as shown in Figure 10-e). It is important to observe that, in this case, further analysis of the second intersection point (2) would not be useful to add other areas, and the sub-algorithm ends. A similar approach is followed to detect the objects involved by the third intersection point (3, Figure 10-f). This time it is not possible to obtain a closed path, and (as shown in Figure 10-g-h-i-l) four poly-lines (P_3 , P_4 , P_5 , and P_6) are found.

4.2 Second Layer Segmentation

Starting from the set of objects (closed regions and poly-lines) identified by the previous module, the aim of the second layer segmentation algorithm is to recognize, on each object, the set of primitives that makes it up. In our context, the term primitive means only two specific geometrical primitives: line and oval (and as particular case every kind of arc and, obviously, the circle).

The reasoning behind the developed algorithm is that every “segment” of a shape can be approximated by the mentioned geometrical primitives. In order to explain the just introduced concept in Figure 12 an example is shown. Deliberately the example regards a messy stroke that improbably can be used to make a symbol. But our intent is to show that the proposed approach is very general, and it can be used on every kind of sketch.

As shown in Figure 11-b, the first searched primitives are the lines (in this case the algorithm detects six lines identified by $L_1..L_6$), the remaining searched primitives are the ovals or, as in this example, the sub-parts (arcs) of different ovals (in this case the algorithm finds, as shown in Figure 11-c, eight arcs $O_1..O_8$).

More formally the discussed algorithm works according to the following main process:

ALGORITHM 4.2.1. *The main step is to analyze every closed-region and poly-line by the following two sequential sub-steps:*

- (i) *line detection: on every object (closed region and/or poly-line) the longer set of points having a linearity property are searched. In this context, the term linearity property is used to indicate a contiguous set of points (having a prefixed minimum length, called L_{fix} , measured in pixels), independently from the orientation, that can be considered belonging to a same prefixed enclosing rectangle (i.e. a oriented rectangle, as shown in Figure 13, having a fixed R_{height}).*
- (ii) *oval (and sub-parts) detection: every set of points discarded by the previous step is an oval (or arc) candidate. For this reason a curvature property, on this set of points, is checked. In this context, the term curvature property is used to indicate a contiguous set of*

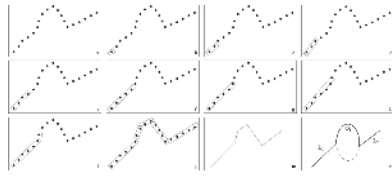


Figure 12: Example of Line and Oval Detection.

points, independently from the orientation, in which the linearity property is true but only on contiguous sub-set of points having a length lower than L_{fix} .

It is important to observe that the main parameters (L_{fix} and R_{height}) can depend on different factors, including: interaction tool, device, pre-processing approach used to "clean" the user sketch, and so on. Indeed, the two steps work cooperatively for "progressive refinements". This means that the algorithm is applied several times until every set of points has found a suitable collocation as line or oval (or part of it).

In order to explain the just mentioned algorithm, in Figure 12 a brief description about the working algorithm is given. In Figure 12-a the set of points representing a stroke is given. As shown, the algorithm starts from the first point (Figure 12-b) and tries to approximate the following points by a prefixed enclosing rectangle (Figure 12-c, Figure 12-d, Figure 12-e, Figure 12-f and Figure 12-g). When there is not an orientation of the enclosing rectangle able to content all the analyzed points (Figure 12-h), the previous enclosing rectangle is considered, and a new enclosing rectangle is created to analyze the remaining points (Figure 12-i). This approach ends when the set of points is entirely analyzed (Figure 12-l). After this step, all the enclosing rectangle are used to approximate the related points with a vectorial line (Figure 12-m). The set of lines which elements have a length lower (or equal) than a fixed threshold (in this example $L_{fix} = 4points$) are approximated with an oval or part of it (Figure 12-n).

Obviously, this kind of reasoning is better applied to real cases, where the strokes are made up from several points. Besides, it has to be taken into account that when the algorithm approximates a set of "short" lines with an oval (or arc) there is the possibility to modify also the elements that surrounds the oval (the left element and the right element, in this case L_1 and L_2). The aim of this refinement step is to provide the best approximation to the set of points that makes up the oval (or arc). The modification of these elements is a very simple task. It consists in the losing of some points and to provide them to the set of points that makes up the oval (or arc). At the end of the developed algorithm a first step toward the vectorization is accomplished. In fact, the following "substitutions" between the found primitives and basic SVG (Scalable Vector Graphics, which is an XML based language) shapes is performed:

- Primitive Line: defined by Line SVG basic shape having the following main attributes: x_1 : the x-axis coordinate of the start of the line (default value is 0), y_1 : the y-axis coordinate of the start of the line (default value is 0), x_2 : the x-axis coordinate of the end of the line (default value is 0), and y_2 : the y-axis coordinate of the end of the line (default value is 0).

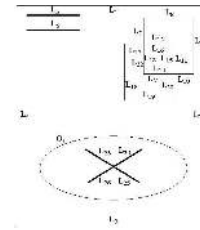


Figure 13: Recognition of: Lines and Ovals.

- Primitive Oval: defined by Ellipse SVG basic shape (and as particular case the circle) having the following main attributes: c_x : the x-axis coordinate of the center of the ellipse (default value is 0), c_y : the y-axis coordinate of the center of the ellipse (default value is 0), r_x (length): the x-axis radius of the ellipse, r_y (length): the y-axis radius of the ellipse.
- Primitive Part of Oval (every kind of Arc): defined by Path SVG basic shape having the following commands: M = moveto, L = lineto, H = horizontal lineto, V = vertical lineto, C = curveto, S = smooth curveto, Q = quadratic Belzier curve, T = smooth quadratic Belzier curveto, A = elliptical Arc, Z = closepath.

Figure 13 shows the recognition of the primitives on the example given in Figure 6. It is possible to observe the generation of some duplicates lines (for example the line: L_{11} and L_{15}) during the algorithmic process, this is correct because when a line belongs to two different objects (as in this case that belongs to two different closed regions) the lines have to be considered separately. In fact, this allows the general working of the proposed approach.

4.3 Constraints Determination

Starting from the set of primitives (lines and/or ovals (or arcs)) that makes up every object (closed region and/or poly-line), the aim of the constraints determination is to detect, for every object, the spatial relationships between them. More specifically, the constraints are searched on each one of the just found vectorial primitives: line, ellipse, and path. The constraints obtained from our recognizer engine can be subdivided according to the following classification:

- Individual Constraints: evaluated on each single vectorial primitive.
- Internal Constraints: evaluated between primitives of the same object.
- External Constraints: evaluated between primitives of different objects.

The following specific measures represent some of the most common adopted constraints:

- Individual Constraints: *orientation* (to indicate the degree orientation of a primitive), *closure* (to indicate the distance between the start point and the end point), etc..
- Internal Constraints: *coincident* (to indicate the geometrical coincidence (in a point) between two primitives), *equalLength* (to indicate an equal length between two primitives), *intersect* (to indicate the geometrical intersection (in one or more points) between

two primitives), *rightAngle* (to identify a right angle (90 degrees) between two lines), etc..

- External Constraints: *contain* (to indicate the geometrical containment of a set of primitives in another one); *vertical alignment* (to identify the vertical alignment of a set primitives with another), etc..

The just mentioned constraints are able to convey detailed relationships on both a single primitive and a set of primitive. Indeed, there are several internal and external constraints that describe same measures, the difference is that the first one works on primitives of a same object, while the second one works on different objects. Every constraint has several attributes that specify the constraint type. For example the individual constraint *orientation* has an attribute to identify the orientation degree, in our context this attribute has seventy-two values to indicate, with a tolerance of step five degrees, the possible orientation of a line. Obviously, if *orientation* is applied to another primitive (e.g. path) it assumes a different meaning (it is used to indicate the concavity and the convexity).

4.4 SketchML Descriptor

Indeed, when the recognition process reaches this module every hard task has been performed. In fact, according to the detected constrains, this module has only to “translate” the just obtained vectorial information (i.e. the SVG basic shapes and the related constrains) in XML language. This set of vectorial information, suitably translated, makes up the main constructs of the SketchML.

The amount of information (i.e. the XML description and related constraints) needed to represent (by SketchML) a simple and single symbol is very large, in Figure 14 a real simplified example of only some individual and internal constraints is given. In particular, the shown example is related to a simple shape representing a square, as the external square (shown in Figure 13) made up from the lines: L_1 , L_2 , L_3 and L_4 . The first kind of shown constraints (individuals) describe only the orientation of each line, while the second kind of shown constraints describe some spatial relationships between the lines. For example, the *GreaterLength* constraint (applied respectively to the arguments L_1 , L_2) describes that the second line (L_2) is greater than the first line (L_1) by about 40 percent.

It is important to highlight, once again, that the XML/SVG structure tends to provide both a suitable way to manage the symbols and a standard approach to increase interoperability between different devices and related applications and services.

5. SKETCH RECOGNITION ENVIRON.

This environment is used to recognize one or more library symbol. In the first case (for example when a user expresses a command) the environment works exactly as the LEE. Another further step (as shown in the next section) is the matching process to map the performed sketch with a related library symbol. In the second case (for example when a user expresses a concept) it is necessary some sort of pre-processing (performed by the context, that is, the loaded library) to recognize (first of all) how many symbols the user has drawn. After this step, on each “candidate” object, the matching process can be performed. To accomplish this task

```

<Individual>
-----
<Orientation item="Line-1" degree="horizontal"/>
<Orientation item="Line-2" degree="vertical"/>
<Orientation item="Line-3" degree="horizontal"/>
<Orientation item="Line-4" degree="vertical"/>
-----
<Internal>
<Internal>
-----
<Coincident first="Line-1" second="Line-2"/>
<GreaterLength first="Line-1" percent="40" second="Line-2"/>
<RightAngle first="Line-1" second="Line-2"/>
<Coincident first="Line-1" second="Line-4"/>
<GreaterLength first="Line-1" percent="40" second="Line-4"/>
<RightAngle first="Line-1" second="Line-4"/>
<Coincident first="Line-2" second="Line-3"/>
<GreaterLength first="Line-2" percent="40" second="Line-3"/>
<RightAngle first="Line-2" second="Line-3"/>
<Coincident first="Line-3" second="Line-2"/>
<GreaterLength first="Line-3" percent="40" second="Line-2"/>
<RightAngle first="Line-3" second="Line-2"/>
-----
<Internal>

```









Figure 14: A Brief SketchML Representation.

all the intersection points (of the sketch performed by the user) are evaluated. After this step the system detects the intersection points that do not have “correspondence” with the intersection points used to build the library, in this way a first evaluation about the number of the sketched objects can be performed. Obviously, these intersection points may be absent, and the same library can be made using symbols without intersection points.

6. MATCHING PROCESS

The final purpose of matching between symbols sketched by the user in the SRE and the symbols represented in a specific library (made up by LEE) is to choose the right element inside the library that provides the suitable interpretation of the sketched symbols. As mentioned, every symbol of the loaded library inside the framework is described through an XML/SVG file. Also the user’s sketch has been represented using XML/SVG file that contains information comparable to the symbols of the library. The matching algorithm performs an exhaustive search of the particular patterns created by the user during sketch activity. Every SVG basic primitive (that is, Line, Ellipse and Path), coming from library or user’s sketch, is represented by a node and a set of relationships (the constraints) between the nodes. Moreover, two or more related basic symbols (for example the containment constraint) are represented by two macro-nodes between which exist a macro-relation (in this case, the *containment*). The matching approach starts from a generic node (of the sketch representation), it performs a matching action with every node of the same kind (such as: line, ellipse or path) presents in the symbol of the loaded library. Obviously, the process appears exponentially complex specially when the first nodes are examined. This happen because the system has more information about the sketch in according to the amount of the analyzed nodes of the sketch. The next step is to get the first (not yet examined) node of the sketch that has to be assigned to another node of the loaded library. To perform this second step it is necessary to exploit the relationships between the previous selected node and the actual selected node. In this way, with the increasing of the examined nodes decrease the disagreeing probability space. The process ends when all the nodes are examined. When this happens several possible configuration will be identified by the system. Each one will have a rank depending on the quality of the accomplished matching. Several configuration

Table 1: Assistive Environment Domain

N° Symbols 154 (Examples)	N° Tracing	Related Vectorization	% of Recognition	% of Misinterpretation
	72		100%	0%
	80		96%	0%
	100		95%	0%
	67		99%	0%
-----	-----	-----	-----	-----
Tot: 11.570	Average 75%	Tot: 11.570	Average 98%	0.5%

will have a very low rank, others will have a suitable ranking profitable to identify the right pattern (i.e. the right symbol).

7. EXPERIMENTAL RESULTS

This section shows the results obtained on a wide application domain designed to meet the needs of heterogeneous assistive environments. In particular, the domain is made up from several simple and complex symbols able to express commands to drive different devices in any context. As shown in Table 1 the application domain was made up from 154 different symbols. Several symbols (e.g. arrow) can have up to eight different orientations (North, East, South, West, Northeast, Southeast, Southwest, Northwest) each of which is considered a different symbol. For each symbol, an average of 75 tracking were performed, for a total of 11.570 tests. The involved population was made up from about 80 people, of these 65% were university students between 25 and 35 years old. The remaining were volunteers (e.g. employees, researchers) with over 56 years old. The population was equally divided between males and females and it had a 15% of left-handed. All people have drawn all symbols according to a given schema in which was highlighted the symbol that had to be drawn and the related number of times.

The percentage of recognition on each symbol is almost total. Indeed, this percentage tends to be slightly lower according to the complexity of the symbol. In fact, for symbols made up from more than 7 strokes some constraints may be ambiguous and misinterpreted. In order to overcome this problem, it is possible to introduce in the framework a more complex personalization concept. This would allow the framework to recognize each symbol according on both the vectorial model created by the framework during the user activity in LEE and some similarity criteria tied to the way in which the user has designed the symbol. Moreover, it is possible to consider a new class of constraints (i.e. control constraints) able to oversee ambiguous potentially situations. The framework has been also tested on more traditional domains (i.e. entity relationship diagram, data flow diagram, electrical schemes, and so on) providing always a total level of reliability. Ultimately it is important to highlight that the framework builds the symbol according to the user drawing. For this reason, for example, the symbols shown in the first and fourth row of the Table 1 are considered different. Indeed, this is a huge advantage for

the personalization of the user libraries. Moreover, if necessary, can always be introduced equivalence criteria between symbols in order to relax the constraints.

8. CONCLUSIONS

The proposed approach provides a pragmatic sketch-based framework able to build and to recognize every kind of 2D graphical library. The key point is that a user, without special knowledge, can create a personalized library simply sketching the wanted symbols. The simple and versatile language used to represent the sketch activity (SketchML) allows the cooperation of the proposed framework with a large number of devices and related applications/services. The framework can be adapted to support also the 3D libraries (adding another category of 3D-oriented constraints).

9. REFERENCES

- [1] C. Alvarado and R. Davis. Dynamically constructed bayes nets for multi-domain sketch understanding. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 33. ACM, 2007.
- [2] L. W. B. Peng, Y. Liu and G. Huang. Sketch recognition based on topological spatial relationship. In *Book Series in LNCS, Structural, Syntactic, and Statistical Pattern Recognition*, volume 3138/2004, pages 434–443. Springer Berlin / Heidelberg, 2004.
- [3] T. H. et al. Free-sketch recognition: putting the chi in sketching. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, pages 3027–3032. ACM, 2008.
- [4] V. M. G. C. G. Casella, V. Deufemia and M. Martelli. An agent-based framework for sketched symbol interpretation. *Journal of Visual Languages and Computing*, 19(2):225–257, 2008.
- [5] T. Hammond and R. Davis. Ladder: a language to describe drawing, display, and editing in sketch recognition. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 27, New York, NY, USA, 2006. ACM.
- [6] E. Jiang and Z. Sun. Hmm-based on-line multi-stroke sketch recognition. *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, 7:4564–4570, 2005.
- [7] G. Sahoo and B. Singh. A new approach to sketch recognition using heuristic. 8(2):102–108, 2008.
- [8] T. Sezgin and R. Davis. Hmm-based efficient sketch recognition. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 281–283, New York, NY, USA, 2005. ACM.
- [9] T. M. Sezgin and R. Davis. Sketch recognition in interspersed drawings using time-based graphical models. *Comput. Graph.*, 32(5):500–510, 2008.
- [10] S. xia Wang, M. tun Gao, and L. hua Qi. Freehand sketching interfaces: early processing for sketch recognition. 4551/2007:161–170, 2007.
- [11] B. Yu and S. Cai. A domain-independent system for sketch recognition. In *GRAPHITE '03: Proc. of the 1st intern. conf. on Computer graph. and interactive tech. in Australia and South East Asia*, pages 141–146, New York, NY, USA, 2003. ACM.