An Agent-Based and Context-Oriented Approach to Symbol Recognition in Diagrammatic Drawings

Giovanni Casella, Vincenzo Deufemia, Viviana Mascardi

An Agent-Based and Context-Oriented Approach to Symbol Recognition in Diagrammatic Drawings

Giovanni Casella
DISI – Università di Genova
Genova, Italy
DMI – Università di Salerno
Fisciano (SA), Italy
casella@disi.unige.it

Vincenzo Deufemia
DMI – Università di Salerno
Via Ponte don Melillo
84084, Fisciano (SA), Italy
deufemia@unisa.it

Viviana Mascardi
DISI – Università di Genova
Via Dodecaneso 35
16146, Genova, Italy
mascardi@disi.unige.it

Abstract

In the last two decades, one new technology, that of agent-based systems, and one emerging research discipline, that of on-line recognition of hand-drawn diagrams, have gained wide attention and consensus. Since the application of the agent technology to disciplines where, traditionally, more standard approaches are adopted, usually leads to valuable and interesting results, we propose an agent-based system for on-line recognition of hand-drawn diagrams. In our system, agents are used 1) to manage the activity of parsers implemented according to the grammar formalism of *Sketch Grammars,* 2) to coordinate themselves in order to provide efficient and precise interpretations of the sketch to the user, and 3) to solve ambiguities by exploiting contextual information.

Keywords: Sketch understanding, agent-based systems, diagram recognition, visual language parsing.

1. Introduction

In the last two decades, one new technology, that of agent-based systems, and one emerging research discipline, that of on-line recognition of hand-drawn diagrams, have received growing attention and consensus.

The AgentLink III Technology Roadmap [15] introduces agent-based systems as:

"*[...] one of the most vibrant and important areas of research and development to have emerged in information technology in the 1990s. Put at its simplest, an agent is a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains.*"

The application of the agent technology in disciplines where, traditionally, more standard approaches were adopted, usually leads to valuable and interesting results. This happened for example to applications in the logistics, transportation, utility management, defense, and e-commerce fields, where commercial companies are heavily investing on agents.

As far as the recognition of hand-drawn diagrams is concerned, we may observe that, although it finds a natural application in a wide range of domains, such as engineering, software design, and architecture, it still remains a particularly difficult task since freehand sketching is an inherently imprecise process. Moreover, the symbols of a sketched diagram can be drawn by using a different stroke-order, -number, and -direction, and the recognition of continuous sketches also involves the activities of segmentation and clustering of the user's strokes at the same time. To make these problems more tractable, many recognition systems work under some assumptions about how the sketches are drawn [20]. Contextual information can be helpful in recognizing hand-drawn symbols, since ambiguities in the sketches can be correctly solved by analyzing the context around the ambiguous parts. In particular, when a recognized symbol is unique to a context then the recognizer may use this symbol to determine the context and thereby resolve pending recognition ambiguities.

In this paper, we propose to apply the agent technology to on-line recognition of hand-drawn diagrams, a domain area where the adoption of agents is still in its early infancy. The approach upon which parsers are based, is the grammar formalism of *Sketch Grammars* for modeling diagrammatic sketch notations and for the automatic generation of the corresponding recognizers [7]. Agents are used to manage the activity of parsers and to coordinate themselves in order to provide efficient and precise interpretations of the sketch to the user.

The paper is organized as follows. Section 2 outlines our recognition approach. Section 3 describes the grammar formalism for sketch languages and the associated parsing technique, and Section 4 describes the multi-agent system built on top of these parsers. Related work and final remarks are discussed in Section 5.

2. The Proposed Recognition System

The architecture of the proposed agent-based sketch recognition system is shown in Fig. 1. The system takes the sequence of *strokes* drawn by the user on the sketch-based interface as its input. A stroke, defined as the locus of the tip of the pen from pen-down to pen-up positions, is represented by a sequence of points. Since the sampling density depends on the sketching speed, a *re-sampling process* is needed to ensure a correct and effective recognition. When the density of raw points is high, re-sampling deletes redundant points in order to

reduce calculation complexity; on the other hand, when the density is low, it adds more points in order to reduce recognition error.

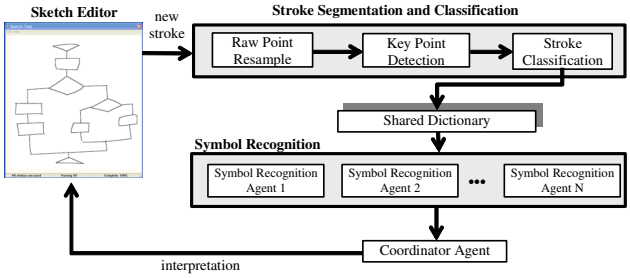

Figure 1. The architecture of the sketch recognition system.

After re-sampling, the *recognition of key points* takes place. A key point is a point that contains the most characterizing geometric features of a sketch. For example, a high curvature point, a tangency point, a corner point and an inflexion point. Key points are likely to be the points that separate the composite sketch into simple strokes, allowing symbols to be drawn with multiple pen strokes, and a single pen stroke to contain multiple symbols. We apply the key point detection algorithm IPAN99 [5].

When key points and strokes have been recognized, the proposed system faces to activity of *classifying the obtained strokes* using fitting algorithms for primitive shapes. In this paper we consider two types of primitive shapes: lines and arcs. We apply the least square fitting for the recognition of lines [8], and the technique proposed in [18] for the elliptical arcs. The classified strokes are stored into a shared dictionary implemented using a blackboard architecture [12].

Intelligent agents are exploited from this stage on. In fact, the *recognition of the domain language symbols* takes place by applying incremental parsers managed by *Symbol Recognition Agents*, (SRAs), to the classified strokes stored into the dictionary. Each SRA is responsible for the recognition of a domain symbol, and faces this task by managing the life cycle of the incremental parsers associated to it (activating, suspending, and killing them), and by exploiting knowledge on the domain context. When a new stroke classification enters the shared dictionary, each SRA checks if it can belong to the symbol it is recognizing. In positive case, it gives the classified stroke in input to its associated parser(s). Contextual information is obtained by cooperating with other SRAs in the system. When a symbol is almost completely recognized, the SRA associates a context value to its interpretation, and sends this information to the *Coordinator Agent* (CA).

The information associated to symbol interpretation allows the CA to solve possible conflicts and to give an interpretation of the sketch drawn so far. The CA is also able to apply heuristics in order to identify symbol interpretations that can be pruned, thus reducing the number of active parsers.

3. Specification and Generation of Symbol Recognizers

In this section we present Sketch Grammars (SkGs, for short) [7], which is the grammar formalism used in the proposed recognition approach for modeling the shape of the domain symbols, and the associated parsing strategy that is a suitable extension of the well-known LR technique [2].

3.1 Sketch Grammars

The idea underlying the formalism is to use grammar productions for clustering pen strokes that constitute input sketches, into shapes of the domain language. Productions have also associated actions that allow designers to specify the display of the recognized shapes, to specify editing gestures, and to define routines for verifying properties on the sketches.

Sketch Grammars represent a direct extension of context-free string grammars, where more general relations other than concatenation are allowed: an SkG *G* can be seen as a context-free string attributed grammar where the productions have the following format:

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

A is a nonterminal symbol, each x_j is a terminal or nonterminal symbol, each p_j is an optional value between 0 and 100 indicating the importance of the shape x_j in the modeled symbol, and each $\mathbf{R_j}$ is a sequence of (spatial and/or temporal) relations $(\langle REL_{j1}^{h1}(t_1),...,REL_{jn}^{hn}(t_n)\rangle)$ with $1 \le k \le n$. $REL_{ji}^{hi}(t_i)$ relates attributes of x_{j+1} with attributes of x_{j-hi}, with $0 \le h_i < j$, by means of a threshold t_i. We denote $REL_1^0(0)$ simply as REL_1.

Act specifies the actions that have to be executed when the production is reduced during the parsing process. These may include a set of rules used to synthesize the values of the attributes of A from those of $x_1, x_2, \ldots, x_m$, a set of display instructions used to display properties of the sketches after the strokes are recognized, etc. Actions are enclosed into the brackets { }. Γ is used to dynamically insert new terminal shapes in the input during the parsing process, enhancing the expressive power of the formalism.

Thus, SkGs specify a sentence by combining symbols with spatial and temporal relations. The idea of associating *importance values* to the shapes of the productions comes from noting that the symbols of a particular domain have different structural complexity. As an example, Use Case Diagrams [17] include *Actor* symbols, which are obtained through the composition of several strokes, and *Participate* symbols, which are simple lines. Hence, importance values can be associated to the complex domain symbols. They allow the generated recognizers to associate a value to the partially recognized symbols, in order to aid the recognition of messy or incomplete symbols. Indeed, the shapes that allow us to distinguish a symbol from the others with a high degree of certainty will have associated a high importance value.

The following production specifies the *Actor* symbol

Actor → Ellipse (45) <$joint_{1\_1}(t_1)$>
Line$_1$(25) < $near(t_2)$, $near^1(t_3)$>
Line$_2$(12) < $joint_{2\_1}(t_4)$, $near^1(t_5)$, $near^2(t_6)$>
Line$_3$(3) <$joint_{2\_1}^2(t_7)$, $rotate^2(135,t_8)$>
Line$_4$(12)<$joint_{2\_1}^3(t_9)$, $rotate^3(135,t_8)$> Line$_5$(3),
{ Actor.attach(1) = Ellipse.attach(1) ∪ Line$_1$.attach(1); }

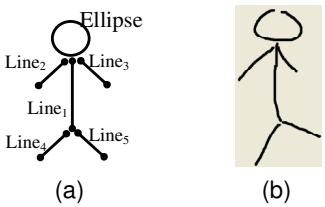

Figure 2. The Actor symbol.

The *Actor* symbol is composed by an ellipse and five lines, as shown in Fig. 2(a) (the attributes are represented with bullets). The non-terminals *Ellipse* and *Line* cluster the single stroke arcs that form an ellipse and the parallel single stroke lines, respectively. The attribute 1 of *Ellipse*, which represents its borderline, is jointed to the attributes 1 of $Line_1$, $Line_2$, and $Line_3$. The latter are rotated with respect to the former of 45 and -45 degrees, respectively. The values $t_1,\ldots,t_9$ specify the error margin in the satisfaction of the relations. The attribute 1 of *Actor* is calculated from the values of the attributes of *Ellipse* and $Line_1$. Finally, the importance values indicate that the head and the body of the *Actor* symbol have a greater weight with respect to the other strokes for discriminating an incomplete actor symbol. The strokes depicted in the sketch of Fig. 1(b), satisfy the relations established by the production rules.

By means of similar rules, it is possible to define *Use Case* symbols (that are ellipses), *Participate* symbols (lines), *Generalize* symbols (arrows), etc.

3.2 Symbol Recognizers

Symbol recognizers try to cluster the classified strokes into symbols of the domain language. The parsing technique extends the approaches proposed in [6]: the parsers scan the input in an incremental and non-sequential way, driven by the spatial relations specified by the grammar productions.

Each symbol recognizer is automatically generated from a sketch grammar modeling the shape of domain symbol [7]. The input to the incremental parser is formed by the stroke classification stored in the shared dictionary, a parse tree, and a graph stack built on the strokes analyzed so far. The parser restructures the parse tree, which represents the recognized strokes, on the base of the new strokes, and updates the graph stack. Each node of the tree has associated a *truthfulness value* and an *importance rate* obtained by combining the accuracy values and by summing the importance values, respectively, associated to the grammar symbols in the graph stack. The output of the symbol recognizer is obtained by analyzing the parse tree and the graph stack, and consists of the quintuple: *name* of the symbol, *attributes* of the symbol for calculating context information, *stroke sequence* analyzed by the parser, which corresponds to the leaves of the parse tree, *truthfulness value* and *importance rate.*

Fig. 3 shows the recognition process of an actor symbol. In particular, from the bottom to the top it shows the incremental editing of the symbol, the primitive shapes (more similar to the last edited stroke) identified by the primitive shape recognizer, and the incremental construction of the parse tree. Note that the third edited stroke is divided in two simple strokes.

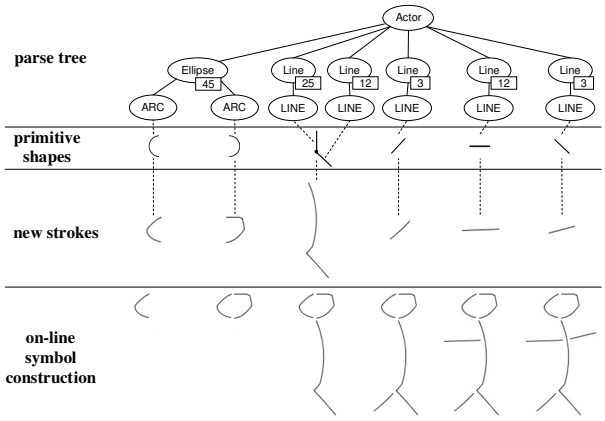

Figure 3. Incremental recognition of an actor symbol.

4. Intelligent Agents for Coordinating and Disambiguating the Recognition Process

An intelligent agent is a software or hardware entity aware of its dynamic and unpredictable environment, able to react to the changes that take place in it, driven by long-term goals, autonomous, and social [13].

If we consider the activity of freehand drawing using computer-aided tools, the "virtual blank sheet" where the user draws represents a dynamic and unpredictable environment. A set of collaborating entities that monitor this virtual environment, react to changes that take place in it (i.e., new strokes drawn by the user), have a complex long term goal (giving an interpretation to what the user is drawing), operate in an autonomous way to reach this goal (no explicit input or suggestions is required to the user), and cooperate to reach their goal (thus exhibiting a social behavior), can be definitely called a multi-agent system, or MAS.

Thus, the adoption of intelligent agents for facing the problem of recognizing freehand drawing sketches seems very appropriate and promising. As already introduced in Section 2, our MAS consists of many instances of Symbol Recognition Agents (SRAs) and of one Coordinator Agent (CA).

4.1 Symbol Recognition Agents

The main goal of each SRA is to recognize domain symbol instances (*Actor*, *Participate*, *UseCase*, *Include*, *Generalize*, etc., in our running example inspired by UML Use Case Diagrams) by collaborating with other SRAs to obtain contextual feedback.

In order to recognize domain symbol instances, each SRA manages the execution of a set of parsers (described in Section 3) for that symbol, and decides when starting, killing or suspending them.

The life cycle of each SRA is characterized by four phases: 1) check the shared dictionary for new interesting strokes; 2) try to recognize a symbol using the new strokes found during the first step; 3) collaborate with other SRAs to obtain feedback on the recognition; and 4) interact with the CA.

Checking the shared dictionary. When a new stroke becomes available in the dictionary, each SRA decides whether the stroke may be interesting for recognizing its domain symbol or not. For example, the SRA that recognizes the *Actor* symbol (Actor SRA) is interested in both arcs and lines, while the *Generalize* SRA is interested only in lines.

Recognizing a symbol. The technical aspects of the recognition process performed by a parser have been outlined in Section 3.2. Here, we consider the behavior of the Actor SRA to exemplify how SRAs use these parsers. Let us suppose that an arc enters the dictionary before an oblique line, and that before the arc entry, the dictionary was empty. When the Actor SRA realizes that there is an arc in the dictionary, it starts a new parser process, with the arc as input. The parser recognizes that the arc might be part of the head of an actor, and updates its recognized symbol. When the line enters the dictionary, the Actor SRA gives it in input to the running parser. In our example, the parser using the arc cannot use the oblique line: according to grammar of the Actor symbol given in Section 3.1, the parser needs that both the actor's head (the arc) and body have been drawn, before being able to attach arms or legs (the oblique line) to it. Thus, a new parser is started with only the oblique line as input. When a parser is fed with a new stroke in input, it also needs to reconsider the other strokes in the dictionary. In fact, it might happen that the usage of a new stroke makes the usage of other pre-existing strokes possible, thanks to production rules that become able to reduce.

It may happen that two parsers are recognizing the same instance of the symbol, although they started from different initial states and applied different production rules. If two parsers reach the same state and are recognizing the same instance of symbol, the SRA kills one of them. Note that the user might draw more instances of the same symbol (for example, two *Actors* related with a *Generalize* symbol). In this case, the mechanism of creating a new parser for each stroke ensures that all the drawn instances will be analyzed and recognized. When one parser reaches a high importance rate in the recognition of a symbol, the symbol is added to the set of the symbols recognized by the SRA.

As an example, let us consider the recognition process shown in Fig. 4. The numbers associated to the strokes denote the temporal sequence of the drawing process. The strokes from 1 to 6 are recognized as the actor *a1* by the Actor SRA. Moreover, stroke 1 is also recognized as the *Use Case* symbol *u1*, whereas the line strokes from 2 to 6 are recognized as the *Participate* symbols *p1*, *p2*, *p3*, *p4*, and *p5*. Strokes 7 and 8 are correctly recognized as the *Participate* symbol *p6* and *UseCase* symbol *u2*, respectively. Finally, stroke 9 is recognized as the *Participate* symbol *p7*, but it is also used with strokes 5 and 6 to recognize the *Generalize* symbol *g1*.

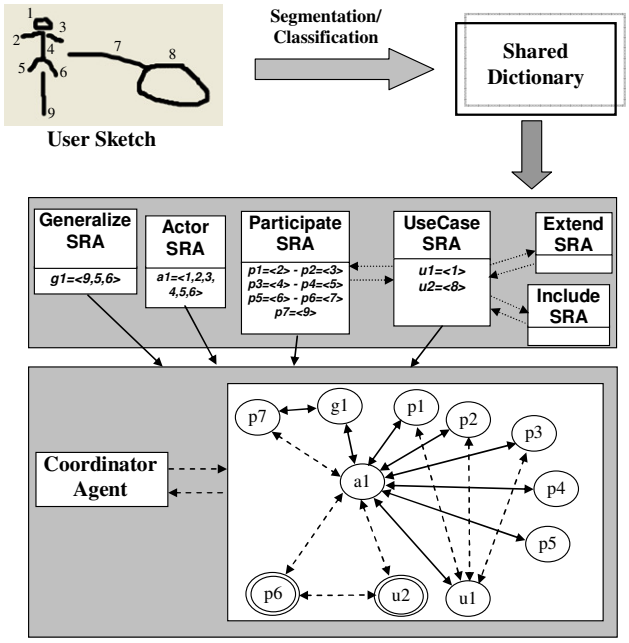

Figure 4. The recognition process of a UML Use Case diagram.

Collaborating with other SRAs. When a symbol is recognized, the SRA starts the collaboration process to obtain context information for the recognized symbol, while the parser goes on with its activity. The collaboration consists of sending a feedback request message containing information about the recognized symbol to all the SRAs that recognize related symbols (and that are known "a priori" by each SRA).

Two domain symbols are *related* if the domain language defines a relation between them. When a SRA receives a feedback request, it checks its set of recognized symbols to give an answer. If it finds a symbol that satisfies the language relationship with the symbol in the feedback request, it sends a positive response to the requester, otherwise it sends a negative response. As an example, in UML Use Case diagrams the *Use Case* symbol is related to *Participate*, *Include* and *Extend* symbols [17]. Thus, when the Use Case SRA in Fig. 4 recognizes *u2*, it sends a feedback request to Actor SRA, Extend SRA, and Include SRA. The first replies with a positive response, since *a1* is correctly related to *u2*, while the others reply with a negative response.

Interacting with the Coordinator Agent. At each editing step, besides updating the information about previously recognized symbol, the SRAs communicate to the CA the information about the new recognized symbols (whose importance rate is higher than a given

threshold) including the set of strokes that form the symbol, its truthfulness value, and the positive feedback collected.

4.2 The Coordination Agent

The CA incrementally analyzes the information received from SRAs and produces an interpretation of the hand-drawn diagram as output. The CA looks for conflicts by checking if there are symbols that share one or more strokes. Conflicts may take place either because a stroke is classified both as a line and as an arc due to the sketch inaccuracy, or because the same stroke, although correctly classified, is used by two SRAs to recognize two different symbols. As an example, in Fig. 4, strokes 5 and 6 are used to recognize both *g1* and *a1*.

In order to support the incremental resolution of conflicts, the CA uses a graph structure to efficiently represent both the information produced by the SRAs, and that obtained during the resolution of the conflicts. In particular, the nodes of the graph correspond to the symbol interpretations provided by the SRAs, whereas the edges can be of two types. The *conflict edges* link conflicting symbols and are labeled with the difference between the truthfulness associated to the symbols (in absolute value), whereas the *feedback edges* link symbols that have produced a positive feedback during their recognition. The conflict between two symbols is solved in favor of the one having the following higher value:

$$cv = w_1\, tr + w_2(\frac{\#rn}{\#n})$$

where *tr* is the truthfulness value of the symbol, *n* is the total number of nodes, *#rn* is the number of nodes without conflicts (*unambiguous symbols*) reachable by following a feedback edge from the symbol, and w_1 and w_2 are values between 0 and 1 that depend on the domain language. In particular, for languages where symbols in the diagrams are involved in many relations with other symbols, w_2 must be greater than w_1, in order to weight the existence of feedback more than the truthfulness of the symbol. Vice versa, for languages with few relations between symbols in diagrams, it is more important to consider the truthfulness associated to the symbol, and thus w_1 must be greater then w_2. Unambiguous symbols are used to solve conflicts because they represent stable and not conflicting elements in the current sketch interpretation.

Conflicts are solved starting from:

1. Those that involve one symbol with feedback from unambiguous symbol(s) (*unambiguous feedback*) and one symbol without unambiguous feedback.
2. Those that involve symbols with higher difference between the number of unambiguous feedback.
3. Those that involve symbols with higher difference of truthfulness value.

This criterion helps in solving the "easiest" conflicts first, in order to obtain new unambiguous symbols that can be used to solve other conflicts.

At the bottom of Fig. 4 it is shown the graph constructed by the CA on the input sketch, where the conflict edges and feedback edges are visualized with continuous arrows and dashed arrows, respectively. Symbol *a1* is in conflict with several symbols (*p1*, *p2*, *p3*, *p4, p5, u1*). The first conflict that is solved is the one between *a1* and *p1*. Indeed, *a1* collected two unambiguous feedback from *p6* and *u2*, while *p1* did not receive unambiguous feedback (the one from *u1* is not unambiguous). Supposing that *a1* has a greater *cv* value than *p1*, *a1* wins the conflict. The conflict resolution goes on and if *a1* becomes an unambiguous symbol (in case it wins all its conflicts), then *p7* gets an unambiguous feedback from *a1* useful to solve its conflict with *g1*.

When a conflict is solved, the graph is updated. When a new (modified, resp.) symbol is communicated to the CA, a new node is added to the graph (the node corresponding to the symbol in the graph is updated, resp.) together with the corresponding conflict and feedback edges. The conflict resolution is applied to that portion of the graph reachable from the new (modified, resp.) node. Thus, the resolution of the conflicts does not involve those parts of the diagram that are not related with the added or modified symbol.

In order to reduce the number of active parsers the CA selects and communicates to the SRAs the ones that can be pruned. Many heuristics can be chosen: for example, pruning could be applied to parsers that have recognized symbols without feedback, and are involved in conflicts with symbols having feedback, or to parsers recognizing symbols whose constituent strokes all belong to another symbol with more positive feedback. For example, in Fig. 4, the *Participate* symbols *p1*, *p2*, *p4*, and *p5* can be pruned since they are sub-pieces of the *Actor* symbol *a1* and they have no positive feedback.

5. Related and Future Work

In the last two decades several approaches have been proposed for the recognition of freehand drawings. Among the most traditional and old systems, we may cite the Rubine recognition engine, a trainable recognizer for single stroke gestures represented by global features and classified according to a linear function of the features [19]. The Electronic Cocktail Napkin (ECN) employs a bottom-up recognition method able to represent ambiguities in the user's sketches, and capable of refining its early interpretations by analyzing the surrounding context [9]. By the observation that in certain domains people draw objects using consistent stroke orderings, Sezgin and Davis have proposed a technique to model and recognize sketches using Hidden Markov Models (HMM) [20]. This approach exploits regularities to perform very efficient segmentation and recognition, but requires each object to be completed before the next one is drawn. Kara and Stahovich [11] present a domain-independent, multi-stroke, trainable shape recognizer that learns new definitions from single prototype examples. The assumptions under which this approach work are that the sketch always includes "marker symbols" easy to recognize, and that the hand-

drawn diagram always consists of shapes linked by arrows. Finally, in [3] Alvarado and Davis present a parsing approach based on dynamically constructed Bayesian networks.

With respect to these systems, our proposal aims at creating a general sketch recognition system that does not rely on any assumption on the drawing style, and is not tailored to any specific domain.

The main originality of our work consists of the exploitation of intelligent agents for the coordination of parsers automatically generated from sketch grammars. Thus, our system is definitely different from all those just discussed. If we come to consider the technology that mainly characterizes our approach, we find that very few approaches are based on it. One of the oldest systems we are aware of, is QuickSet, a suite of agents for multimodal human-computer communication [4]. A very similar, but more recent, agent-based multimodal system is Demo, described in [10]. In [1], Achten and Jessurun discuss how graphic unit recognition in drawings can take place using a multi-agent systems approach, where singular agents may specialize in graphic unit-recognition, and multi-agent systems can address problems of ambiguity through negotiation mechanisms. In [16], Mackenzie and Alechina propose an agent-based technique for the classification and understanding of child-like sketches of animals, using a live pen-based input device. Finally, in [14] Juchmes and Leclercq describe EsQUIsE, an interactive tool for free-hand sketches to support early architectural design.

When we compare our proposal with those using the agent technology, we find that the main difference lies in the intended usage domain of the system, that is very specific for all the implemented systems. The only general-purpose view is provided by Achten and Jessurun that, however, do not propose a concrete MAS architecture, but just analyze the feasibility of adopting multi-agent techniques to sketch recognition.

The usage domain of our system comprises any visual language that can be modeled by a sketch grammar. In this paper we have used UML Use Case Diagrams to exemplify the functionality of our system, but we could use animal forms, architectural sketches, or Gannt charts as well, provided that we had parsers for them. Luckily, an approach for automatically generating parsers from visual grammars has been proposed by one of the authors [6], and our system has been designed to be fully compliant with these automatically-generated parsers. The internal behavior of the agents in the system and the interactions among them remain unchanged, no matter which parsers are managed by the SRAs.

Another difference between the related agent-based approaches and ours, lies in the technique used to recognize each single stroke, and each symbol from a set of strokes, since we extensively use automatically generated parsers.

The main limitation of our approach, that also drives our current and future work, is the lack of experimental results. Among the other agent-based systems considered in this section, it seems that a prototype exists for all of them, apart from the general proposal in [1]. We are currently implementing a prototype of our system using the multi-agent platform JADE (http://jade.tilab.com/).

References

[1] H.H. Achten, and A.J. Jessurun, "An Agent Framework for Recognition of Graphic Units in Drawings", *Proc. of eCAADe'02*, 2002, pp. 246-253.

[2] Aho, A.V., R. Sethi, and J.D. Ullman, *Compilers Principles, Techniques, and Tools*. Addison-Wesley, 1987.

[3] C. Alvarado, and R. Davis, "Dynamically Constructed Bayes Nets for Multi-Domain Sketch Understanding", *Proc. of IJCAI'05*, 2005, pp. 1407-1412.

[4] P. R. Cohen, M. Johnston, D. McGee, I. Smith, J. Pittman, L. Chen, and J. Clow, "Multimodal interaction for distributed interactive simulation", in *Proc. of IAAI'97*, 1997.

[5] D. Chetverikov and Z. Szabo, "A Simple and Efficient Algorithm for Detection of High Curvature Points in Planar Curves", *Proc. the APRG Workshop*, 1999, pp. 175-184.

[6] G. Costagliola, V. Deufemia, G. Polese, M. Risi, "A Parsing Technique for Sketch Recognition Systems", *Proc. of IEEE Symposium VL/HCC'04*, Italy, 2004, pp. 19-26.

[7] G. Costagliola, V. Deufemia, M. Risi, "Sketch Grammars: A Formalism for Describing and Recognizing Diagrammatic Sketch Languages", *Proc. of ICDAR'05*, Korea, 2005, IEEE Press, pp. 1226-1230.

[8] Duda, R.O., and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley Press, New York, 1973.

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

[10] E. Kaiser, D. Demirdjian, A. Gruenstein, X. Li, J. Niekrasz, M. Wesson, and S. Kumar, "Demo: A Multimodal Learning Interface for Sketch, Speak and Point Creation of a Schedule Chart", *Proc. of ICMI'04*, 2004, pp. 329-330.

[11] L.B. Kara and T.F. Stahovich, "Hierarchical Parsing and Recognition of Hand-sketched Diagrams", *Proc. of UIST'04*, ACM Press, 2004, pp. 13-22.

[12] B. Hayes-Roth, "A Blackboard Architecture for Control", *Artificial Intelligence*, 26(3), 1985, pp. 251-321.

[13] N. R. Jennings, K. P. Sycara, and M. Wooldridge A Roadmap of Agent Research and Development, *Autonomous Agents and Multi-Agent Systems Journal* 1(1), 1998, pp 7-36.

[14] R. Juchmes and P. Leclercq, "A Multi-Agent System for the Interpretation of Architectural Sketches", in *Proc. of 2004 Eurographics Workshop on Sketch-Based Interfaces and Modeling*, Grenoble, France, pp. 53-61.

[15] M. Luck, P. McBurney, O. Shehory, et al. "Agent Technology: Computing as Interaction – A Roadmap for Agent-Based Computing", AgentLink III, 2005.

[16] G. Mackenzie and N. Alechina, "Classifying Sketches of animals using an Agent-Based System", *Proc. of CAIP'03,* 2003, LNCS 2756, pp. 521-529.

[17] Object Management Group. *UML version 2.0*, 2005, http://www.omg.org/docs/formal/05-07-04.pdf

[18] M. Pilu, A. Fitzgibbon, and R. Fisher, "Direct Least-Square Fitting of Ellipses", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21(5), 1999, pp. 476-480.

[19] D. Rubine, "Specifying Gestures by Example", *Computer Graphics* 25(4), 1991, pp. 329-337.

[20] T.M. Sezgin and R.Davis, "HMM-based Efficient Sketch Recognition", *Proc. of IUI'05*, 2005, ACM Press, pp. 281-283.