ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

Journal of Visual Languages and Computing

▮ (▮▮▮▮) ▮▮▮–▮▮▮

Journal of Visual Languages & Computing

www.elsevier.com/locate/jvlc

An agent-based framework for sketched symbol interpretation

Giovanni Casella[a,b], Vincenzo Deufemia[b,*], Viviana Mascardi[a], Gennaro Costagliola[b], Maurizio Martelli[a]

[a]*Dipartimento di Informatica e Scienze dell'Informazione—Università di Genova Via Dodecaneso 35, 16146, Genova, Italy*

[b]*Dipartimento di Matematica e Informatica—Università di Salerno, Via Ponte don Melillo, 84084 Fisciano (SA), Italy*

Received 16 March 2007; received in revised form 12 April 2007; accepted 24 April 2007

Abstract

Recognizing hand-sketched symbols is a definitely complex problem. The input drawings are often intrinsically ambiguous, and require context to be interpreted in a correct way. Many existing sketch recognition systems avoid this problem by recognizing single segments or simple geometric shapes in a stroke. However, for a recognition system to be effective and precise, context must be exploited, and both the simplifications on the sketch features, and the constraints under which recognition may take place, must be reduced to the minimum.

In this paper, we present an agent-based framework for sketched symbol interpretation that heavily exploits contextual information for ambiguity resolution. Agents manage the activity of low-level hand-drawn symbol recognizers, that may be heterogeneous for better adapting to the characteristics of each symbol to be recognized, and coordinate themselves in order to exchange contextual information, thus leading to an efficient and precise interpretation of sketches. We also present AgentSketch, a multi-domain sketch recognition system implemented according to the proposed framework. A first experimental evaluation has been performed on the domain of UML Use Case Diagrams to verify the effectiveness of the proposed approach.

Keywords: Sketch understanding; Agent-based systems; Diagram recognition; Visual language parsing

*Corresponding author.

E-mail addresses: casella@disi.unige.it (G. Casella), deufemia@unisa.it (V. Deufemia), mascardi@disi.unige.it (V. Mascardi), gcostagliola@unisa.it (G. Costagliola), martelli@disi.unige.it (M. Martelli).

1045-926X/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2007.04.002

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

1. Introduction

The recognition of hand-sketched symbols is a very active research field, since it finds a natural application in a wide range of domains, such as engineering, medicine, and architecture [1–7]. However, it is a particularly difficult task since the symbols can be drawn by using a different stroke-order, -number, and -direction. The difficulties in the recognition process are often made harder by the lack of precision and by the presence of ambiguities in messy hand-drawn sketches [8,9]. In fact, hand-sketched symbols are imprecise in nature: corners are not always sharp, lines are not perfectly straight, and curves are not necessarily smooth.

Usually, hand-drawn sketches consist of shapes whose meaning depends heavily on context. For example, a single line fragment could constitute the side of a box, or a connector between boxes, and its role could be disambiguated only by looking at neighboring fragments. This means that, when a recognized symbol is unique to a context, then the recognizer may exploit this information to determine the context and thereby resolve pending recognition ambiguities. The context can also be used to recover from low-level interpretation errors by reclassifying low-level shapes, obtaining significantly reduced recognition errors [10].

Besides this, the literature describes many proposals where sketch interpretation is carried out by applying, to any symbol in the sketch, the same recognition approach. To the best of our knowledge, no framework exists that allows the adoption of different techniques for recognizing different symbols. Nonetheless, this is a very useful feature for an effective recognition process, since each symbol shows its own peculiar characteristics, which makes a particular recognition technique more or less suitable for it.

In this paper, we present an agent-based framework for sketched symbol interpretation. The reasoning process performed by the intelligent agents devoted to symbol recognition (*Symbol Recognition Agents*, SRA for short) and to the correct interpretation of the sketch (*Sketch Interpretation Agent*, SIA for short), is based on the knowledge about the domain context, which is used for disambiguating the recognized symbols. SRAs exchange contextual information by cooperating with other SRAs in the system. The contextual information obtained in this way is sent to the SIA that solves possible conflicts and gives an interpretation of the sketch drawn so far. At the lowest level of our framework, the symbols of the domain language are recognized by applying suitable *hand-drawn symbol recognizers* (HDSRs, for short) to the input strokes. The execution of these HDSRs is coordinated by SRAs. In spite of the differences among the existing HDSRs, several of them could be profitably integrated into our system. As long as there is one SRA that correctly integrates a set of HDSRs by managing their execution as well as data conversion issues, the actual implementation of the HDSRs and the approach to recognition that they adopt *do not matter*. For this reason, our framework has the potential to *seamlessly integrate symbols* recognized by *heterogeneous* HDSRs.

Moreover, we describe the AgentSketch system, an implemented instance of the general framework, which can be applied to a variety of domains by just integrating the proper symbol recognizers. We have applied AgentSketch to the domain of UML Use Case Diagrams and performed a preliminary evaluation study. The obtained results have highlighted the effectiveness of the proposed context-based recognition approach.

The paper is organized as follows. Section 2 illustrates three scenarios that, despite of their differences, would all benefit from a multi-agent approach to sketch recognition.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

Section 3 explains why we claim that the multi-agent approach is suitable to sketch recognition, and outlines the agent-based general framework we propose. Sections from 4 to 6 describe the three main components of our general framework, namely HDSRs, SRAs, and SIA, respectively. Section 7 describes the AgentSketch system and its preliminary evaluation. The related work is discussed in Section 8. Finally, conclusions and further research are presented in Section 9.

The preliminary version of this paper was published in the proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, 2006 [11].

2. Motivating examples

To motivate the agent-based approach to hand-drawn sketch recognition discussed in Section 3, we analyze three types of sketches taken from three different domains: Use Case Diagrams (software engineering domain), ancient Egyptian hieroglyphs (hand-written patterns domain), and anatomic sketches (medical domain). We have chosen these types of sketches because they raise different challenges that could be suitably faced following an approach based on intelligent software agents. In the sequel of this section we will discuss each of them; we will pay more attention to the Use Case Diagrams, since they are used throughout the paper to show the design, implementation and use of our system.

2.1. Use Case Diagrams

Description. The first type of sketch is taken from the software engineering area, where sketching is mainly used to represent software design ideas in terms of diagrams. It is a common practice for the designers to model software systems by using the UML diagrammatic notation [12], the de-facto standard for engineering Object-Oriented software. In particular, during the requirement analysis phase, designers draw Use Case Diagrams to describe the available functionalities and the main usage scenarios of an application. Such diagrams are composed of *use cases* (represented by ovals), *actors* (represented by stick figures), and connectors among them (Fig. 1).

There is only one type of relationship that may occur between *actors* and *use cases*; it is visualized like a solid line, named *communication* link, and means that an *actor* participates to a *use case*. Instead, four types of relationships between *use cases* are supported by UML: *communication*, *inclusion* (visualized as a dashed arrow from the including to the included *use case*, with label "include"), *extension* (a dashed arrow from the extending to the extended *use case*, with label "extend"), and *generalization* (a solid line ending in a hollow triangle drawn from the specialized to the more general *use case*). The only type of relationships that may hold among *actors* is generalization.

Being able to correctly interpret hand-drawn Use Case Diagrams would allow software engineers to converge more quickly towards an agreement in situations like brainstorming

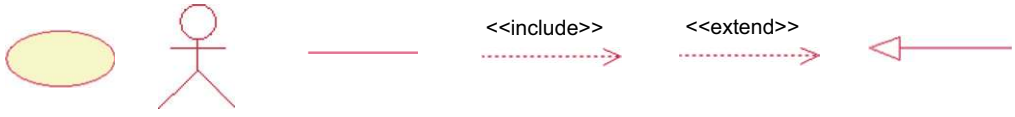


Fig. 1. Symbols used in a Use Case Diagram.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

meetings and teleconferences, where manually sketching the ideas that emerge during the conversation are easier, more convenient and rapid than drawing them by means of a CASE tool. An example of UML Use Case Diagram is shown in Fig. 2. Fig. 2(a) shows a diagram drawn using a UML editor, and modeling the functionality of account creation for the Administrator of a blog. Fig. 2(b) shows a hand-drawn version of the same diagram, without considering textual descriptions whose recognition is out of the scope of this paper. Note that *extend* and *include* symbols have been drawn with solid lines for simplifying the drawing process.

Challenges raised by Use Case Diagrams recognition. The recognition of the hand-drawn Use Case Diagram in Fig. 2(b) presents several challenges. In fact, users should be able to draw without indicating where one symbol ends and the next one begins, and should be free to draw a symbol with any number of strokes and in any order. Indeed, *use case* symbols might be drawn as a single pen stroke, or as two or more separated strokes. Alternatively, a single pen stroke might contain multiple shapes, as the *communication* symbol and a *use case* in Fig. 2(b). Moreover, another problematic condition occurs when two different symbols present similar parts. Considering the example of Fig. 2(b), how is it possible to distinguish the communication links from a line composing the actor or another relationship symbols? Finally, a last problem is due to the on-line processing of the sketch. In fact, new strokes become available while the user is drawing, and these newly disclosed strokes may change the interpretation previously given to a symbol. Thus, solving ambiguities must undergo an incremental approach, where any new information coming from the user is exploited for tuning the sketch interpretation given so far.

The variation in drawing style may be managed for example by an ink parsing process that establishes which strokes are part of which shapes by grouping and segmenting the user's strokes into clusters of intended symbols [13]. It may be also useful to integrate, within the system, symbol recognizers implemented following different approaches; in fact, recognizing a symbol composed by many strokes, like the *actor*, might benefit from the adoption of techniques different than those used to recognize a *communication* symbol, usually made by just one stroke.

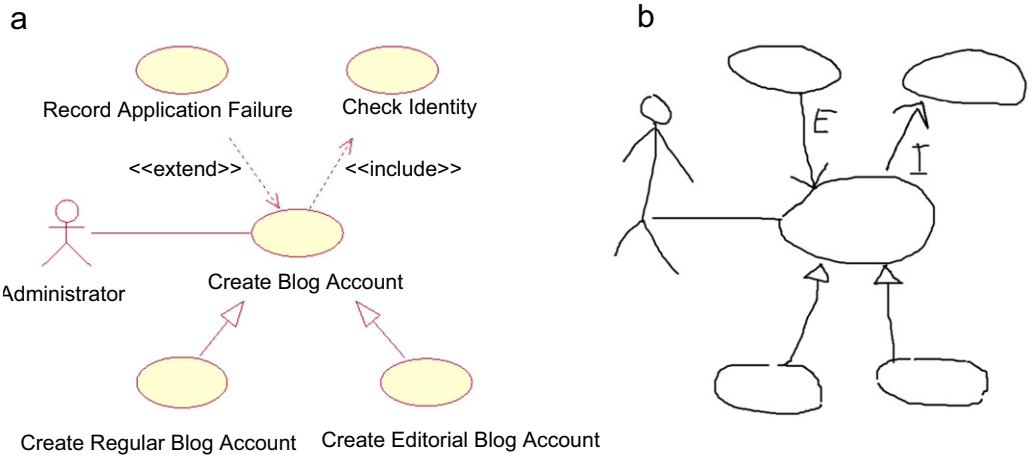


Fig. 2. A Use Case Diagram drawn with a UML editor (a) and using a sketch-based interface (b).

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

The ambiguities due to the possible membership of one stroke to more than one symbol, can be solved by analyzing the objects surrounding the ambiguous parts, i.e., the context around them. As soon as the context changes, or new strokes become available for recognizing new symbols or for correcting previously given interpretations, the system must be able to use this new information in an incremental way, namely, without re-interpreting portions of the sketch whose interpretation does not raise any conflict, and working only on the ambiguous portions.

2.2. Ancient egyptian hieroglyphs

Description. Ancient Egyptian hieroglyphs consist of more than 2000 logographic, alphabetic, and ideographic elements. Logographs may represent either a word or a morpheme (a meaningful unit of language). The logographic approach stands in contrast to other writing systems, such as alphabets, where each symbol (letter) primarily represents a sound or a combination of sounds. Ideographic elements are similar to (sometimes used as synonyms of) logographic ones, but they represent pure ideas rather than words of morphemes. Examples of hieroglyphs are shown in Fig. 3.

Challenges raised by hieroglyphs recognition. The challenges raised by recognizing Egyptian hieroglyphs, as well as similar pattern-based writing systems such as the ancient logographic languages of Near East, India, China, and Central America, are almost different from those raised by diagrammatic sketch recognition. In fact, although the hieroglyph, like a symbol in a diagrammatic language, can be divided into a set of primitives shapes with relationships among them, there is no global "sketch" in the written document, and the recognition process works on each symbol of the language (each hieroglyph, in this case) in an independent manner. Besides this, although all ancient hieroglyphs are, of course, "hand-written", the hand-writing process does not take place on-line. It is in fact reasonable to assume that hieroglyphs are supplied to the recognition system as scanned images, thus making recognition an off-line process. This means that primitive patterns composing each hieroglyph are not produced by a sketch editor, but by a static image segmentation system. Thus, no information on the temporal succession of pattern drawing may be exploited for helping the recognition. Also, the same hieroglyph may have a different appearance in different documents, due to differences in the period, place and style of the document composition: the recognition process must be able to recognize the same symbol despite of its different representations and to the different supports where it has been engraved or painted (see Fig. 4).

Finally, the set of known hieroglyphs is still evolving: as new historical documents are discovered by the archaeologists, new hieroglyphs are classified and interpreted. Thus, the system should be able to easily integrate new hieroglyph recognizers without requiring any substantial change to its code and behavior.

Based on the challenges identified so far, a system for the recognition of ancient Egyptian hieroglyphs should be able to work off-line, taking scanned images as input; do

Fig. 3. Some Egyptian hieroglyphs (from Wikipedia, http://en.wikipedia.org/wiki/Egyptian_hieroglyph).

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

not make assumption on the temporal succession of drawn primitive patterns; be open to the insertion of new hieroglyph recognizers as new hieroglyphs are discovered; be flexible enough to recognize the same symbol also when represented in different fashions, like the system described in [14] that uses Fuzzy Hierarchical Attributed Graphs for recognition of hieroglyphs.

2.3. Anatomical sketches

Description. As observed by Haddawy et al. in [3], drawing anatomical sketches is a common practice in medicine. Physicians use sketches when taking notes in patient records and for conveying diagnoses and treatments to patients, and medical students use them for supporting reasoning about clinical problems in individual and group problem solving.

The top right box of Fig. 5 shows a template of the external view of lung, while in the left part of the figure, the first column shows the sketches drawn by different students, the second column is the physician's segmentation of the sketch, and the last column is the segmentation performed by UNAS, the automatic system described in [3]. By looking at the sketches, it is clear that they show deep differences in line style and width; also, they may either miss strokes or contain more strokes than expected.

Challenges raised by anatomic sketches recognition. Understanding an anatomical sketch requires the ability to recognize what anatomical structure has been sketched and from

Fig. 4. Different representations of the scarab symbol.

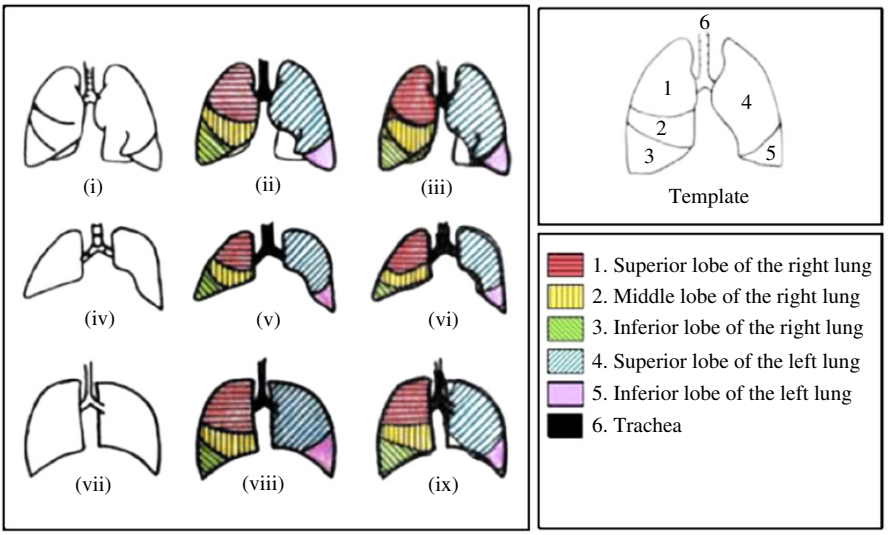


Fig. 5. A template and three hand sketches of lung's external view (courtesy of Haddawy et al. [3]).

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

what view (e.g., parietal view of the brain), as well as to identify the anatomical parts and their locations in the sketch (e.g., parts of the brain), even if they have not been explicitly drawn. The lowest level objects of the sketches are not geometric primitives, such as lines, rather they are symbols that must be recognizable in isolation.

Thus, an automatic sketch recognizer should be able to exploit symbol recognizers that perform an image-based analysis in order to avoid many problematic issues such as segmentation and feature extraction. Symbol recognizers should be tolerant of over stroking, missing and extra pen strokes, variations in line style, and variations in drawing order, since anatomical sketches are usually characterized by all these features.

In this domain, an automatic sketch recognition system might be used both on-line and off-line. Its main requirement would be, once again, the ability to integrate in a seamless way heterogeneous symbol recognizers, since, due to their different complexity, the sketches of some anatomical organs may be better recognized using different techniques.

2.4. Features required by a recognition system for facing the above challenges

If we consider the challenges arising in the three above scenarios, the features that an automatic sketch recognizer should have in order to face them may be summarized in being able to:

- exploit contextual information (when the graphical language of the sketch allows it) for correctly interpreting symbols;
- coordinate the behavior of the symbol recognizers in such a way to detect and solve conflicting interpretations of symbols;
- integrate in a seamless way heterogeneous symbol recognizers in order to exploit different techniques for recognizing different symbols starting from primitive shapes;
- add new symbol recognizers as soon as they become available, without needing to change any other component of the system;
- work both on-line and off-line;
- according to the working modality (on-line vs. off-line), either do or do not exploit information on the temporal succession of drawn primitive patterns;
- receive input from different kinds of devices and in different formats.

We claim that a *multi-agent approach to sketch recognition* may help in designing and implementing a system having the above features. The next section introduces agents and multi-agent systems, and discusses the reasons upon which our claim grounds.

3. The multi-agent approach to sketch recognition

The AgentLink III Technology Roadmap [15] defines an agent as:

> "*a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains*."

According to [16], agents should be

1. *autonomous*: they should operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

2. *responsive*: they should perceive their environment and respond in a timely fashion to changes that occur in it;
3. *pro-active*: they should not simply act in response to their environment, but should exhibit opportunistic, goal-directed behavior and take the initiative where appropriate;
4. *social*: they should be able to interact, when appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities.

Another characterizing feature of agents is *situatedness:* the agent receives sensory input from its environment and it can perform actions which change the environment in some way [17].

As far as sociality is concerned, it is now widely recognized that interaction is probably the most important single characteristic of nowadays' complex software. Two good reasons for agents to interact and eventually cooperate, are to solve conflicts [18,19], and to disambiguate the interpretation of objects in some domain [20].

A multi-agent system, or MAS for short, is a system composed by many interacting agents. In a MAS, each agent has incomplete information or capabilities for solving the problem, thus each agent has a limited viewpoint, there is no global system control, data is decentralized, and computation is asynchronous. Also, due to the dynamicity and unpredictability of scenarios where agents live, MASs are *open to change*. This means that the topology of a MAS cannot be fixed a priori, but it dynamically changes as agents enter and leave the MAS.

If we keep the above features in mind while considering the problem of recognizing hand-drawn sketches in the three scenarios described in Section 2, we soon realize that an architecture based on agents might be a proper solution.

In fact, when drawing takes place as an on-line process, the "virtual blank sheet" where the user draws, represents a dynamic and unpredictable environment. Then, an "entity" devoted to recognizing the sketch drawn by the user must be situated in it, in order to properly perceive the user's actions. Also, this entity must react to changes that take place in the virtual blank sheet, i.e., new strokes drawn by the user. Situatedness and reactivity are not crucial issues if the system processes the input in an off-line way, like when recognizing an Egyptian hieroglyph stored in an image file.

However, despite the on-line vs. off-line processing modality, entities working for recognizing the sketch must have a complex long term goal, i.e., giving a correct interpretation to what the user is drawing or to what the image represents, and must operate in an autonomous way to reach this goal, since no explicit input or suggestions must be required to the user of the system. Although each single entity may be able to perform some task useful for recognizing the sketch (for example, recognizing one specific symbol of the language with a certain degree of confidence), by working alone it cannot easily resolve ambiguities ("Is this symbol an arrow or a line?"), and conflicts ("In order to recognize my symbol, I am using a primitive shape that is also used by another entity; to which symbol does the shape really belong?"). Thus, a social behavior is required to reach the final goal of each entity, which consists in overcoming conflicts and ambiguities, and providing the right interpretation of the sketch to the user. Finally, since the number of graphic elements likely to be recognized is almost unlimited, a modular software architecture is necessary so that entities can be connected or disconnected with the least possible repercussions on the system [21].

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

In the end, each "entity" must be responsive, pro-active, situated, autonomous, and social, and must live in an open environment. In other words, each entity must be an intelligent agent living in a MAS.

Our proposal exploits the suitability of an agent-oriented conceptualization of the hand-drawn sketch recognition problem, and the advantages of an agent-oriented approach to engineering complex systems [22,23], for modeling and designing the agent-based framework depicted in Fig. 6. An implemented instance of this framework is discussed in Section 7 and demonstrates, in the domain of diagrammatic sketch recognition, the feasibility of our approach. In this section, we provide a high-level overview of the general framework, whereas in Section 7 we will discuss all the details of the implemented instance.

Our framework is composed by four kinds of agents:

Interface Agent. It represents an interface between the agent-based framework and the generic "Input Suppliers" that are not included inside the framework. The nature of these input suppliers may vary according to the type of sketch to be interpreted and to the drawing process (on-line vs. off-line). For example, they might be editors suitable for on-line drawing of diagrammatic and anatomic sketches, as well as interfaces that allow the user to select an existing image to be interpreted (useful for an off-line recognition process like that of hieroglyphs interpretation). The Interface Agent informs the "Sketch Interpretation Agent" (SIA) and the "Input Pre-Processing Agent" (that, in turns, informs the "Symbol Recognition Agents") about the nature of the recognition process (off-line or on-line) and converts the information produced by the input suppliers into a suitable format for these agents. It sends each new available piece of input (or the whole input, in case of an off-line recognition process) to the Input Pre-Processing Agent, and interacts with the SIA for sending the sketch interpretation requests to it, and for delivering its answer to the user. While for on-line recognition the user can request the sketch interpretation several times as he/she is drawing, for off-line recognition the sketch interpretation is requested by the user, or automatically by the input supplier, just one time when all the input has been acquired. For any new input supplier to be linked to the framework, a new "stub" inside the Interface Agent must be created ad-hoc. For example,

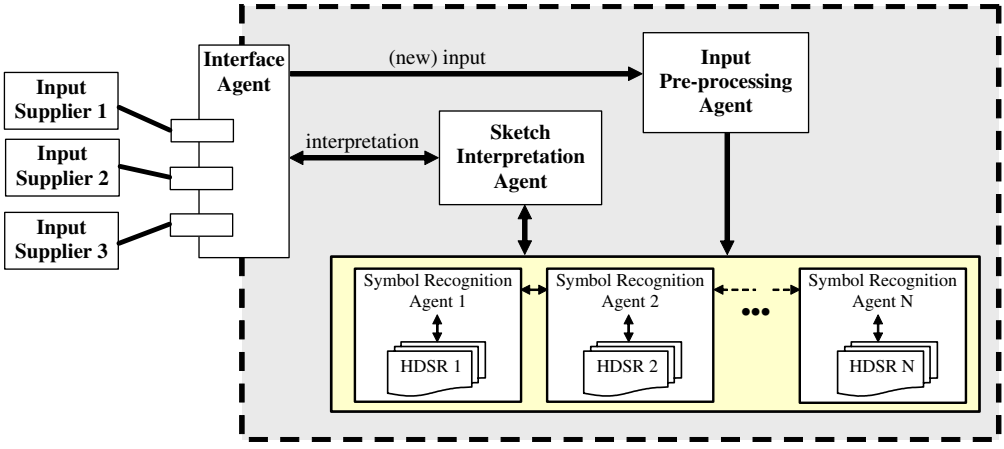


Fig. 6. The agent-based framework for sketch recognition.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

the implemented instance of the Interface Agent discussed in Section 7 provides a stub for an on-line editor that takes advantage of Java Swing components and Satin [24].

Input pre-processing agent. It processes the input received from the Interface Agent and sends the obtained results to the "Symbol Recognition Agents" (SRAs) described in the following, using a format compliant with the recognition approach they apply. For example, if the input to pre-process is a Use Case Diagram sketched on-the-fly, the main activity of this agent will be to segment and classify the user strokes into a sequence of domain independent primitive shapes. It will receive from the Interface Agent the sequence of strokes drawn by the user, and send their classification results, together with their attributes, to the SRAs. In the case of the hieroglyphs recognition problem, the pre-processing agent will behave in a different way: it will be devoted to identify and isolate each single hieroglyph within the sketch, and send it to one or more SRAs, according to some strategy based on the hieroglyph's features.

Symbol Recognition Agents: Each SRA is devoted to recognize a particular symbol of the domain. Moreover SRAs may collaborate with other SRAs in order to apply context knowledge to the symbols they are recognizing, and with the SIA that deals with the sketch interpretation activity. In our running example based on UML Use Case Diagrams, there will be an SRA devoted to recognize "*generalize*" symbols by managing the execution of the related "hand-drawn symbol recognizers" (HDSRs), and by collaborating with other SRAs to obtain contextual feedback. Each SRA tries to recognize a domain symbol by applying suitable HDSRs to the stroke classifications produced by the Input Pre-Processing Agent. The Input Pre-Processing Agent sends a stroke classification to an SRA only if the SRA may use it to recognize new domain symbols (i.e., strokes classified as ellipses will not be sent to the SRA devoted to recognize the *generalize* symbol, which does not include ellipses).

When a new symbol has been recognized by the HDSR, the corresponding SRA starts the collaboration process with other SRAs for obtaining contextual information for the recognized symbol. The collaboration consists of sending a feedback request message containing the attributes of the recognized symbol to all the SRAs that recognize related symbols (that are known "a priori" by each SRA). When an SRA receives a feedback request message, it checks its set of recognized symbols to give a response. 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. For example, due to the rules that govern the definition of well-formed UML Use Case Diagrams, an SRA that recognizes an *actor* symbol should ask a feedback to the SRA that recognizes *communication* symbols, since *actors* always participate to *use cases* to which they are connected via a *communication* symbol.

Sketch Interpretation Agent: The SIA provides the correct interpretation either of the sketch drawn so far (in case of an on-line drawing process) or of the entire sketch (in case of an off-line recognition process) to the Interface Agent. In particular, it analyzes the information received from SRAs and solves conflicts between symbols that might arise. When all the conflicts have been solved, the SIA proposes the sketch interpretation to the user, interacting with the Interface Agent. The SIA looks for conflicts by checking if there are symbols that share one or more strokes. For example, conflicts may take place either because a stroke is classified as two different shapes (for example, 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. By solving the "easiest" conflicts first (namely, those conflicts where it is easier to identify the right interpretation), the SIA

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

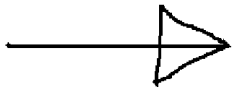
is able to free some symbols from the conflicts they were involved in. These symbols are used for solving other conflicts, and the process goes on until there are no more conflicts left. The SIA also applies some heuristics to select a set of unlikely symbol interpretations, which are pruned by the recognized symbol set of the SRAs improving their recognition efficiency.

Hand-drawn symbol recognizers: HDSRs are not agents since they lack most of the agent-characterizing features. They are just software modules managed by SRAs. Many approaches have been proposed for recognizing free-hand drawings [25–30]. Each of these proposals has the potential of being integrated into our system in spite of the fact that they differ one from another under several aspects, ranging from the identification of the shape of the symbols to the approach used to construct them. As we have already pointed out, as long as there is one SRA that correctly integrates HDSRs by managing their execution as well as data conversion issues, the actual implementation of the HDSRs and the approach to recognition that they adopt do not matter. The framework that we propose may seamlessly integrate symbols that have been recognized by heterogeneous HDSRs managed by ad-hoc SRAs.

4. Hand-drawn symbol recognizers

In this section, we describe the main features of three HDSRs that can be exploited by SRAs to identify the symbols of a given domain: LADDER [25], Sketch Grammars [27], and CALI [28].

LADDER. In LADDER, the symbol recognition is performed using the rule-based system Jess [31]. In particular, for each symbol of the domain, a Jess rule is automatically generated from a LADDER structural shape description, which mainly contains information on the shape of the symbol, but may include other information helpful to the recognition process, such as stroke order or stroke direction. As an example, the LADDER description and the generated Jess rule for the visualized *generalize* symbol of UML Use Case Diagrams are described in the boxes below.



```
define shape GeneralizeCheck
description
"An arrow with a triangle-shaped head"
components
    Line shaft
    Line head1
    Line head2
    Line head3
constraints
    coincident shaft.p1 head1.p1
    coincident shaft.p1 head2.p1
    coincident head1.p1 head2.p1
    equalLength head1 head2
    acuteMeet head1 shaft
    acuteMeet shaft head2
    coincident head3.p1 head1.p2
    coincident head3.p2 head2.p2
aliases
    Point head shaft.p1
    Point tail shaft.p2
```

```
(defrule GeneralizeCheck
;; get four lines
?f0 <- (Line  ?shaft   \$?shaft_list)
?f1 <- (Line  ?head1   \$?head1_list)
?f2 <- (Line  ?head2   \$?head2_list)
?f3 <- (Line  ?head3   \$?head3_list)
;; make sure lines are unique
(test (uniquefields \$?shaft_list \$?head1_list))
(test (uniquefields \$?shaft_list \$?head2_list))
(test (uniquefields \$?shaft_list \$?head3_list))
(test (uniquefields \$?head1_list \$?head2_list))
(test (uniquefields \$?head1_list \$?head3_list))
(test (uniquefields \$?head2_list \$?head3_list))
;; get accessible components of each line
(Line ?shaft ?shaft_p1 ?shaft_p2 ?shaft_length ?shaft_acc)
(Line ?head1 ?head1_p1 ?head1_p2 ?head1_midpoint ?head1_length ?head1_acc)
(Line ?head2 ?head2_p1 ?head2_p2 ?head2_midpoint ?head2_length ?head2_acc)
(Line ?head3 ?head3_p1 ?head3_p2 ?head3_midpoint ?head3_length ?head3_acc)
;;test constraints
(test (perpendicular ?shaft ?head1))
(test (coincident ?shaft ?head1  midpoint))
(test (coincident ?head1_p2 ?head2_p1))
(test (coincident ?head2_p2 ?head3_p1))
(test (coincident ?head3_p2 ?head1_p1))
(test (equalLength ?head1 ?head2))
(test (acuteMeet ?head1 ?head2))
(test (acuteMeet ?head2 ?head3))
(test (acuteMeet ?head3 ?head1))
=> ;; Generalize symbol found
;; add symbol to recognized symbols
   (bind ?acc (computeaccuracy ?shaft_acc ?head1_acc ?head2_acc ?head3_acc))
   (assert (Generalize (head ?shaft_p1) (tail ?shaft_p2) (accuracy ?acc))))
```

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

The LADDER rule defines a *generalize* shape as a list of four lines from which the shape is built plus the geometric constraints defining the relationships on these elements. The generated Jess rule, like all Jess rules, is composed of two parts. The part on the left of the "⇒" symbol contains the name of the rule ("GeneralizeCheck") and the preconditions that enable the rule to fire, namely: getting four lines, making sure that the four lines are unique, getting the components of each line, and finally checking that the lines' components meet the topological and geometric constraints that allow an arrow to be composed with them. The part on the right of the "⇒" symbol, defines what to do when the precondition is met; in this case, the symbol, together with its constituent parts and its accuracy, computed from the accuracy values produced by the primitive shape recognizers, are given in output. Thus, when in the working memory of the HDSR there are four lines that respect the precondition of the rule, the rule is fired and a *generalize* symbol is recognized.

The shapes described with LADDER must be diagrammatic or iconic since they have to be drawn using a predefined set of primitive shapes and composed using a predefined set of constraints.

Sketch Grammars: Sketch Grammars (SkGs) represent a direct extension of string grammars, where more general relations other than concatenation are allowed [27]. The symbol recognizers automatically generated from SkGs try to cluster stroke interpretations into symbols of the domain language. The parsing technique extends the approaches proposed in [32]: the parsers scan the input in an incremental and non-sequential way, driven by the spatial relations specified by the grammar productions.

An SkG G can be seen as a context-free string attributed grammar where the productions have the following format:

$$A_{\Gamma} \rightarrow x_1 \mathbf{R_1} x_2 \mathbf{R_2} \ldots x_{m-1} \mathbf{R_{m-1}} x_m, Act,$$

A is a nonterminal symbol, each x_j is a terminal or nonterminal symbol, and each $\mathbf{R_j}$ is a sequence of spatial and/or temporal relations [27]. *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 $x_1, x_2, \ldots, x_m$. 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.

To go on with our running example based on UML Use Case Diagrams recognition, the Actor SRA might manage an "Actor HDSR" implemented using SkG. This HDSR would use the following production to recognize the *Actor* symbol:

$$\begin{aligned}
&\text{Actor} \rightarrow \text{Ellipse} \ \langle joint_{1_1}(t_1) \rangle \\
&\qquad \text{Line}_1 \langle near(t_2),\ near(t_3),\ rotate(45, t_4) \rangle \\
&\qquad \text{Line}_2 \langle joint_{2_1}(t_5),\ near^1(t_5),\ near^2(t_6),\ rotate^1(-45, t_4) \rangle \\
&\qquad \text{Line}_3 \ \langle joint^2_{2_1}(t_7),\ rotate^2(135, t_4) \rangle \\
&\qquad \text{Line}_4 \ \langle joint^3_{2_1}(t_9),\ rotate^3(-135, t_4) \rangle \ \text{Line}_5, \\
&\{\text{Actor.attach}(1) = \text{Ellipse.attach}(1) \cup \text{Line}_1\text{.attach}(1); \\
&\quad \text{Actor.accuracy} = \text{ComputeAccuracy}();\}
\end{aligned}$$

The *Actor* symbol is composed of an ellipse and five lines, as shown in Fig. 7 (the attributes are represented with bullets). The non-terminals *Ellipse* and *Line* cluster the

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

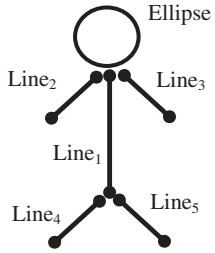


Fig. 7. The Actor Symbol.

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°, respectively. The values $t_1,\ldots,t_9$ specify the error margin in the satisfaction of the relations. Finally, the attribute 1 of *Actor* is calculated from the values of the attributes of *Ellipse* and $Line_1$, and the accuracy of *Actor* is computed by the *ComputeAccuracy* function, which combines the accuracy of the strokes forming the sketch and of their spatial relations.

CALI. CALI is a system for recognizing multi-stroke geometric shapes based on a naïve Bayesian classifier [28]. It is able to identify shapes of different sizes and rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. It detects not only the most common shapes in drawing such as triangles, lines, rectangles, circles, diamonds and ellipses, using multiple strokes, but also other useful shapes such as arrows, crossing lines or wavy lines. A training process has to be performed for adding new shapes to the set of symbols to be recognized.

The shapes are recognized by statistical analysis of various ratios of values such as the convex hull around the area designated as containing a single shape, the largest triangle and largest quadrilateral that can be inscribed within the hull, the smallest area enclosing rectangle that can be fitted around the shape. For handling ambiguities naturally, fuzzy logic is used to associate degrees of certainty to recognized shapes. Thus, the recognizer works by looking up values of specific features in fuzzy sets associated to each shape and command. This process yields a list of plausible shapes ordered by degree of certainty. Nevertheless, the filters applied during the recognition are ineffective on ambiguous shapes such as pentagon and hexagon.

In order to provide on-line recognition of sketches, the systems constructed on top of CALI submit the strokes drawn by the user to the recognizer when the user's pauses are longer than a given time between strokes [33].

5. Symbol recognition agent

An SRA is an autonomous software agent able to control an HDSR and to cooperate with other agents in order to recognize hand-drawn symbols. The SRA behavior is mainly affected by received messages, as highlighted in the state diagram shown in Fig. 8.

Initializing, terminating and reading new messages: In the *Initializing* state each SRA reads the MAS configuration in order to correctly initialize itself. In particular, the SRA learns how to communicate (i.e., address, supported communication protocols, languages,

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

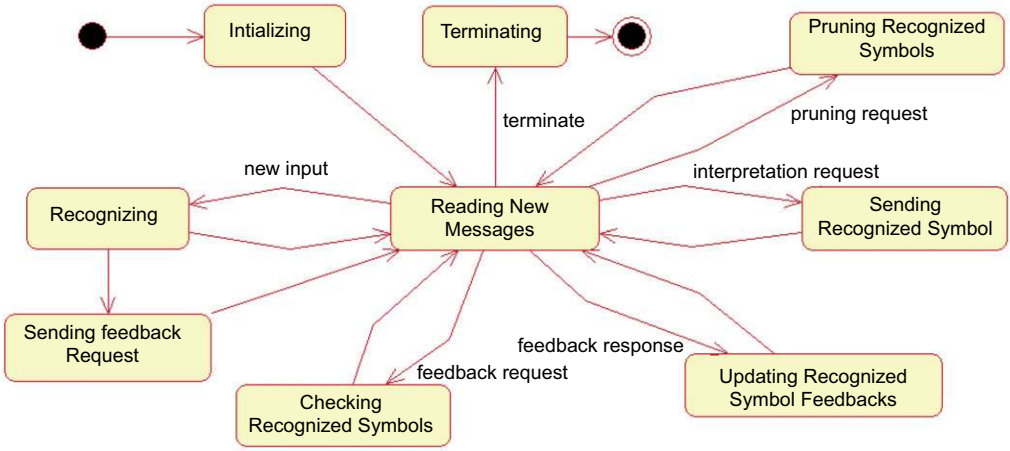


Fig. 8. The state diagram describing the behavior of Symbol Recognition Agents.

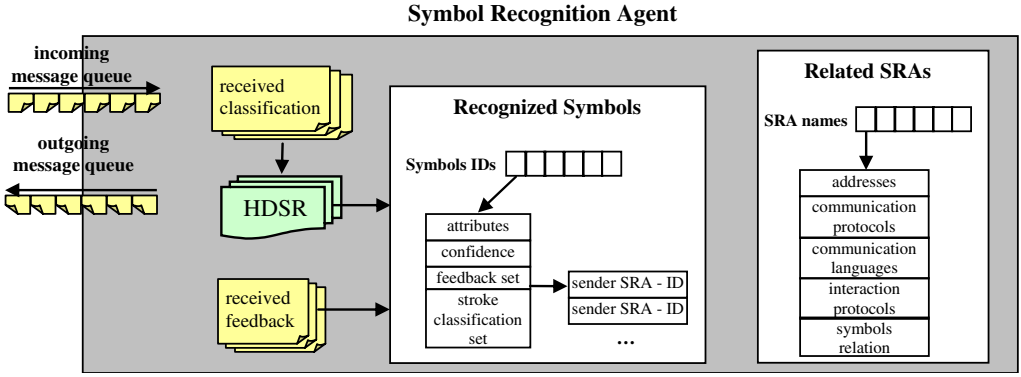


Fig. 9. SRA internal structure.

interaction protocols, and so on) with SRAs that recognize related symbols. This information is stored by the SRAs in the *related SRAs* data structure as shown in Fig. 9. Moreover, the SRA loads some parameters to correctly initialize its HDSR. The operations performed to initialize an HDSR depend on the particular HDSR considered. For example, if the SRA uses a HDSR based on Jess, then suitable rules must be loaded in the Jess engine during the initialization stage, while if the HDSR is based on SkGs, then the parser must be instantiated with the proper rules, and if a CALI HDSR is used, the training samples for the domain symbol must be provided. For other kinds of HDSRs, suitable operations are carried out.

Once the initialization phase has been completed, the SRA goes in the *Reading New Messages* state where it just waits for receiving a message. A *terminate* message is sent to all the SRAs by the Interface Agent if the user stops the execution of the external input supplier, be it a sketch editor or another kind of device. When the *terminate* message is received, each SRA moves to the *Terminating* state, releases the allocated resources and

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

terminates. When an SRA is in a state different from *Reading New Messages*, any new message is stored in the incoming message queue and served when the SRA comes back to the *Reading New Messages* state. On the other hand, when the SRA is in the *Reading New Messages* state and no message is available, then the agent stays idle in a waiting mode.

Recognizing: When a message containing a new input is received from the Input pre-processing agent, the SRA moves from the *Reading New Messages* state to the *Recognizing* state. With the first message the Input pre-processing agent also informs the SRA of the sketch recognition mode (on-line or off-line). The SRA uses this information to correctly configure its HDSR. In the *Recognizing* state, for each received message, the SRA builds a representation of the received data suitable for its HDSR (i.e., a *fact* for a Jess based HDSR, a new token symbol for the sketch parser for an HDSR based on SkGs, etc.). Then the SRA tries to recognize a new domain symbol giving the new available information in input to its HDSR. If a new domain symbol is recognized, then the HDSR outputs the strokes used to recognize it and the symbol attributes (e.g., starting point, ending point, radius, etc.). For each recognized symbol the SRAs compute a confidence value which represents a sort of probability that the symbol has been correctly recognized. The confidence values obtained by the HDSRs require a normalization process since HDSRs can output values belonging to different ranges. The recognized symbols and their features are stored by the SRAs in the set of recognized symbols in a way that they can be easily retrieved using their IDs, as shown in Fig. 9. If a new symbol is recognized when the SRA is in the *Recognizing* state, then it moves to the *Sending Feedback Request* state, else it moves back to the *Reading New Messages* state.

Sending feedback request: When the SRA recognizes a new symbol it sends a feedback request to all the SRAs that recognize related symbols. The technical details necessary to send the message are contained in the *related SRAs* set. The feedback request message contains the attributes of the recognized symbol and its ID. SRAs always include in feedback request and response messages their name (unique in the MAS) and the identifiers of the involved symbols in order to better trace related recognized symbols. The information on related recognized symbols will be used by the SIA to solve conflicts and to prune misleading interpretations.

Checking recognized symbols: When a feedback request is received, the SRA checks its recognized symbols set to verify if the symbol in the feedback request message is related through a given relationship with one or more recognized symbols. If some relation holds, the SRA sends a positive response to the requester, otherwise sends a negative response. As shown in Fig. 9, a feedback vector is associated to each recognized symbol in the recognized symbol set for containing all its related recognized symbols. If the SRA is able to give a positive feedback response then the SRA updates the feedback vector associated with the symbols involved in the positive feedback response.

Updating recognized symbols feedbacks: When a positive feedback response is received, the SRA updates the feedback vector of the symbol that started the feedback request process.

Sending recognized symbol: When the Interface Agent sends an interpretation request to the SIA, it forwards the request to all the SRAs. Then, each SRA sends to the SIA the symbols it was able to recognize together with the set of input drawings used to recognize it, the symbol's confidence, and the symbol's feedback vector.

Pruning recognized symbols: HDSRs could produce wrong symbol interpretations, mainly due to input inaccuracy. In particular, the input components could be not correctly

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

classified or could be used to recognize more than one symbol. For example, if we consider Use Case Diagrams, a line belonging to an *actor* might be also used by the communication SRA to recognize a *communication* symbol. The SIA, as described in the following, is responsible to detect and solve this kind of errors. For on-line recognition, when the SIA terminates the interpretation process, it sends to all the SRAs a message containing the list of all the misrecognized symbols. When this message is received the SRA goes in the *Pruning Recognized Symbols* state in which it updates its recognized symbols set. In particular it deletes all the wrong symbols and updates the feedback vectors of the symbols with feedbacks from the wrong symbols. Moreover, to improve the efficiency of the on-line recognition process, if the recognition process is stroke-based, the SRAs may perform a pruning operation on the unrecognized strokes. As an example, the arrow symbol is often drawn with a single stroke (as the *include* symbol of Fig. 2(b)), which is segmented in four lines, three of which are used in the recognition of the arrow and one remains as unprocessed input. The elimination of these strokes allows the SRAs to improve the performance of the recognition process. However, the selection of the strokes to be removed is not a trivial task and can depend from the domain language.

Considering our running example based on UML Use Case Diagrams, Fig. 10 shows the incremental editing of a diagram and the recognition process performed by SRAs having associated HDSRs constructed with Jess rules. At each edited symbol the figure shows the result of input pre-processing, the symbol recognized by each SRA, the exchanged messages and the obtained feedbacks. In particular, the drawings in the first column show the new edited strokes in black, whereas the previous one are in gray; the numbers associated to the strokes denote the temporal sequence of the drawing process. The second column shows the result of the pre-processing on the new edited strokes. This process consists in the segmentation of the strokes (the identified key points[1] are visualized with a dot) and in the classification of the substrokes obtained. We consider three primitive shapes in the classification process: line, arc, and ellipse. The label associated with the strokes indicates the result of the classification process: L, A, and E denote a line, an arc, and an ellipse classification, respectively. The third column shows for each symbol to be recognized (*Actor, Communication, Use Case, Generalize, Extend, Include*) an appropriate SRA, which includes an appropriate HDSR, and the messages exchanged to compute feedbacks. The recognized symbols are listed on the right side, whereas the solid arrows show the messages producing positive feedbacks while the dashed arrows show those producing negative feedbacks. The recognized symbols that have obtained a positive feedback are in boldface.

When the strokes from 1 to 4 in Fig. 10(a) are drawn, the strokes 3 and 4 are segmented in two substrokes, and then classified in an ellipse $E1$ and five lines ($L_1,\ldots,L_5$). Note that stroke 3 is classified both as a line and an arc, and that the first one has a greater accuracy value. The HDSR associated with the Actor SRA recognizes these primitive shapes as an *actor* symbol. Moreover, stroke 1 is also recognized as the *Use Case* symbol u_1 by the HDSR associated with the Use Case SRA, whereas the line interpretations from L_1 to L_5 are recognized as the *communication* symbols c_1, c_2, c_3, c_4, and c_5 by the HDSR associated with the Communication SRA. As previously described, when an SRA recognizes a symbol it starts to collaborate with SRAs recognizing related symbols for obtaining contextual information. In Use Case Diagrams, the *Use Case* symbol is related to

[1]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.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

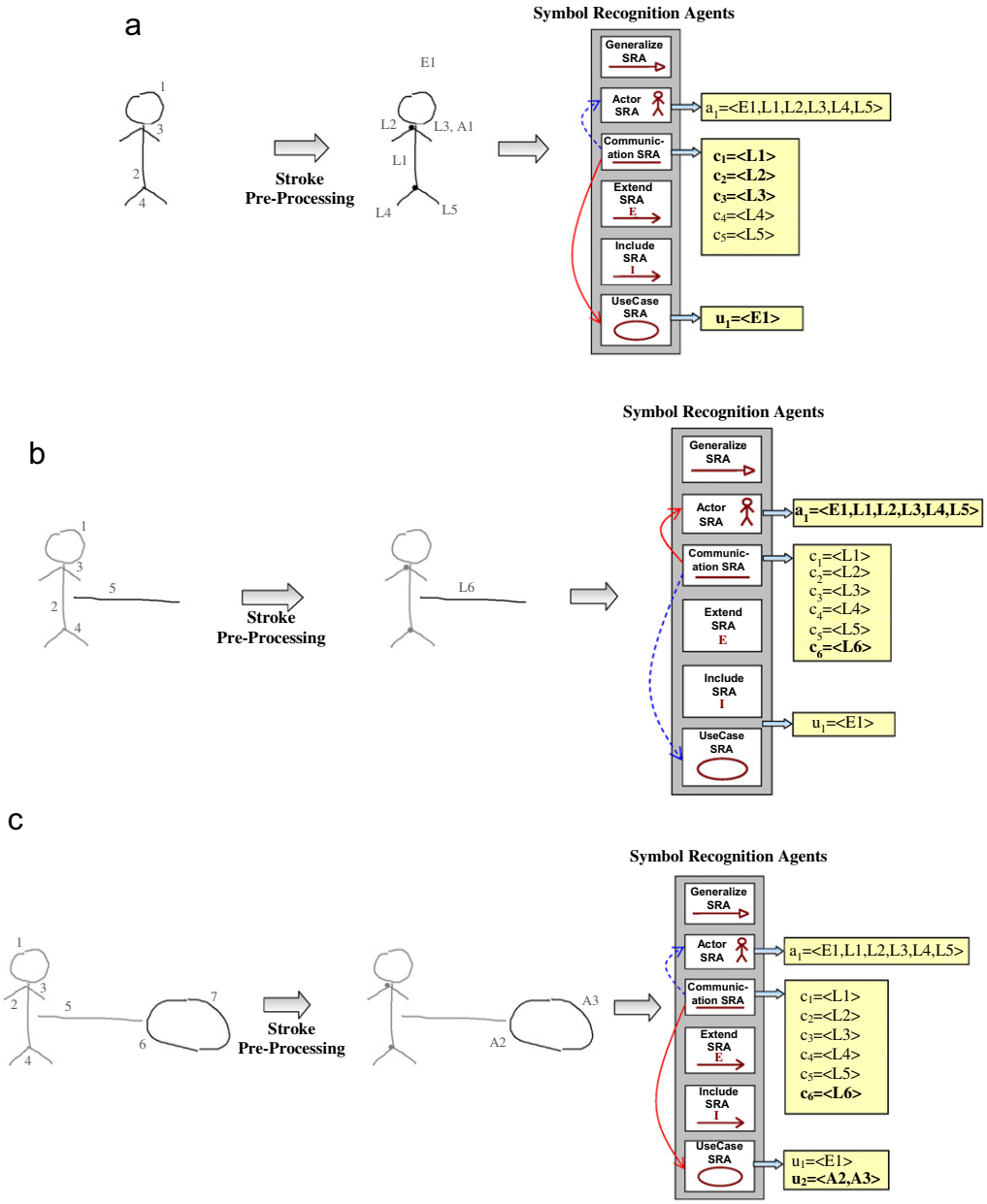


Fig. 10. The recognition of the symbols of a Use Case Diagram by SRAs.

communication, *generalize*, *include* and *extend*. Thus, when the Use Case SRA recognizes u_1, it sends a feedback request to Communication SRA, Generalize SRA, Extend SRA, and Include SRA. All of them reply with a negative response since no other symbol has been recognized yet. These exchanged messages are not shown in the figure. When the Communication SRA recognizes $c_1, \ldots, c_5$, it sends a feedback request to the Use Case and

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

Actor SRAs. The first replies with a positive response to c_1, c_2, c_3, since u_1 is correctly related to them, while the second replies with a negative response. Finally, when the Actor SRA recognizes a_1, it sends a feedback request to the Communication and Generalize SRAs, which reply with a negative response.

When stroke 5 in Fig. 10(b) is drawn, it is classified as a primitive shape line (L_6) and used by the Communication SRA to recognize c_6. Once recognized c_6, the Communication SRA sends a feedback request to Use Case SRA and Actor SRA, and the latter replies with a positive response.

Finally, strokes 6 and 7 in Fig. 10(c) are classified as two arcs, and recognized by the Use Case SRA as symbol u_2. As done previously, the Use Case SRA sends a feedback request to the related SRAs, and receives a positive reply from the Communication SRA since u_2 is related to c_5.

6. The Sketch Interpretation Agent

The goal of the SIA is to give, every time the user requests it, an interpretation of the sketch drawn so far. In order to build the sketch interpretation the SIA requests the information they have computed to the SRAs. In particular, it receives the set of recognized symbols and their collected feedback, which are exploited to obtain the correct sketch interpretation (detecting wrongly recognized symbols).

The SIA behavior is represented by the state diagram in Fig. 11 and described in the following:

Initializing, terminating and waiting for interpretation requests: In the *Initializing* state the SIA initializes itself and reads some configuration files to learn how to communicate with SRAs (i.e., address, supported communication protocols, languages, interaction protocols and so on). The SRAs information are stored in the SRA table as shown in Fig. 12.

When the SIA is in the *Waiting for Interpretation Request* state, it just puts itself in waiting until a request message is received. On the other side if a sketch interpretation request is received while the SIA is not in the *Waiting for Interpretation Request* state then the message is stored in the incoming message queue.

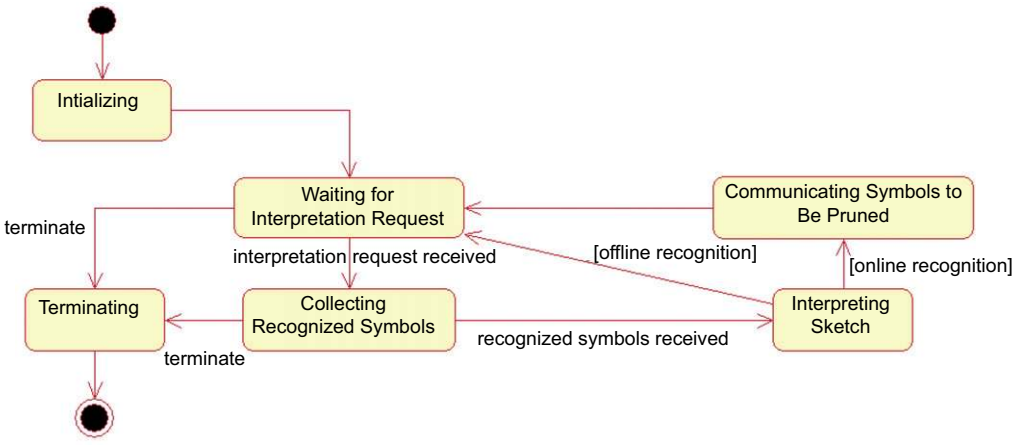


Fig. 11. The state diagram describing the behavior of SIA.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

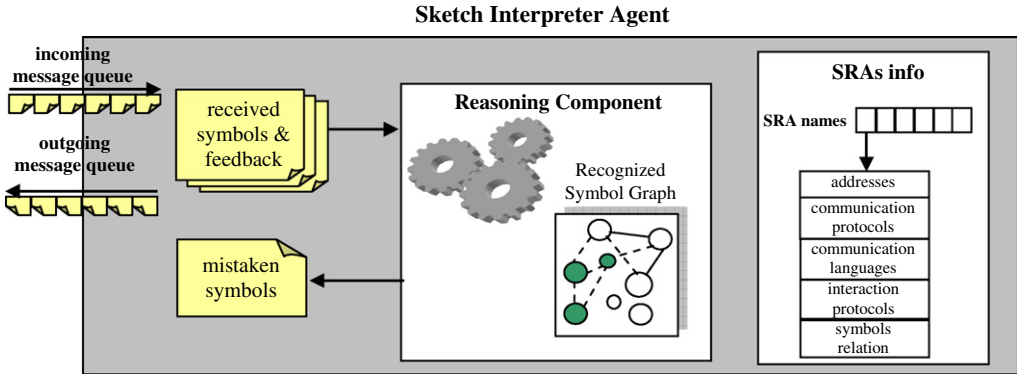


Fig. 12. The SIA internal structure.

If a terminate message is received (it is sent by the Interface Agent when the user closes the editor), independently from its state, the SIA transits to the *Terminating state*, releases the allocated resources and terminates.

Collecting Recognized Symbols: When a sketch interpretation request message is received the SIA transits from the *Waiting for Sketch Interpretation Request* state to the *Collecting Recognized Symbols* state. Here, the SIA sends an interpretation request message to all the SRAs and collects their responses. Using this information the SIA builds the recognized symbol graph. Each node of the graph has associated the symbol that represents with some information, such as the SRA that has recognized the symbol, the symbol confidence, the strokes belonging to the symbol. The graph edges can be of two types: the *conflict edges* link conflicting symbols, whereas the *feedback edges* link the symbols that have produced a positive feedback during their recognition. A symbol without conflicts is named *unambiguous symbol*. The SIA is able to build the graph thanks to the information sent by the SRAs as described in the previous section. When all the SRA responses have been received and the graph has been built the SIA transits to the *Interpreting Sketch* state.

Interpreting Sketch: In this state the SIA looks for conflicts by checking the recognized symbol graph. The SIA goal, in this state, is to solve all conflicts by finding the wrongly recognized symbols, and, consequently, to delete all the conflict edges in the graph. This task is accomplished by the *Reasoning Component*.

The conflict between two symbols is solved in favor of the one having the following higher *truthful* value:

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

where *conf* is the confidence value of the symbol, $\#n$ is the total number of nodes, $\#rn$ is the number of *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 whose diagrams may have symbols involved in many relations with each other, w_2 must be greater than w_1, in order to weight the existence of feedback more than the accuracy of the symbol. Vice versa, for languages with few relations between symbols, it is more important to consider the accuracy associated to the symbol, and thus

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

w_1 must be greater than 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 their numbers of unambiguous feedbacks.
3. Those that involve symbols with higher difference of their accuracy values.

This criterion helps in solving the "easiest" conflicts first, in order to obtain new unambiguous symbols that can be used to solve other conflicts. When a conflict is solved, the graph is updated deleting the node associated to the loosing symbol.

In addition to solve conflicts, the SIA might apply heuristics dependent from the domain language for removing some symbol interpretations. For example, if there are two symbols s_1 and s_2 such that s_1 graphically includes s_2 (e.g., in Use Case Diagrams *actor* includes a *Use Case* symbol), then the SIA can solve all conflicts that arise between these two symbols a priori.

When the conflict resolution phase terminates the SIA sends the sketch interpretation to the Interface Manager that interacts with the Editor to show the sketch interpretation to the user.

When the conflict resolution ends the SIA behavior depends on the drawing process nature (on-line or off-line) communicated by the Interface Manager with the sketch interpretation request. For on-line recognition, the SIA stores the graph before going in the *Communicating symbols to be pruned* state. In this way, the next time the SIA has to interpret the input sketch it only updates the stored graph and solves the new arising conflicts, avoiding to solve the same conflicts more than one time. For off-line recognition, the SIA just goes in the "Waiting for Interpretation Request Message" where it will receive a terminating message by the Interface Manager.

Communicating symbols to be pruned: In this state the SIA selects and communicates to the SRAs the recognized symbols to be pruned. Many heuristics can be chosen: for example, pruning could be applied to HDSRs that have recognized symbols without feedback, and are involved in conflicts with symbols having feedback, or to HDSRs recognizing symbols whose constituent strokes all belong to another symbol with more positive feedback, and so on. The choice of the heuristics to be applied also depends from the diagrammatic language.

As an example, if the user requests the interpretation of the sketch in Fig. 10(c) the SIA constructs the graph in Fig. 13 using the symbols communicated by the SRAs. In the figure solid lines represent conflict edges, dashed lines represent feedback edges, while filled nodes represent unambiguous symbols.

The *actor* symbol a_1 is in conflict with several symbols (the *communications* c_1, c_2, c_3, c_4, c_5, and the *Use Case* u_1). The first conflict that is solved is the one between a_1 and c_1. Indeed, a_1 collected one unambiguous feedback from c_6, while c_1 did not receive unambiguous feedback (the one from u_1 is not unambiguous). Supposing that a_1 has a greater truthful value than c_1, a_1 wins the conflict. The conflict resolution goes on and since a_1 wins all its conflicts, it becomes an unambiguous symbol providing unambiguous feedback.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

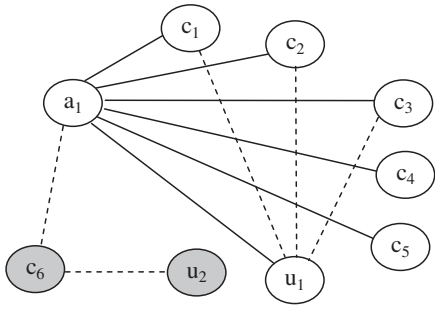


Fig. 13. The graph constructed by the SIA on the Use Case Diagram of Fig. 10(c).

7. The AgentSketch system

AgentSketch is a system for recognizing freely hand-drawn diagrammatic sketches exploiting the multi-agent approach presented in Section 3. It supports both on-line and off-line recognition mode and can be applied to a variety of domains by providing the suitable recognizers. In the following, we describe the implementation details and then we present an experimental evaluation of the system on the domain of Use Case Diagrams.

7.1. Implementation

The architecture of AgentSketch, that instantiates the general one presented in Fig. 6, for recognizing UML Use Case Diagrams, is shown in Fig. 14. AgentSketch has been implemented on top of the Jade agent-based platform [34] using Java 1.5. Agents in AgentSketch communicate by exchanging messages encoded in FIPA-ACL messages [35], a communication language natively supported by Jade. FIPA-ACL specifies both the message fields and the message "performatives" (communicative acts such as requesting, informing, making a proposal, accepting or refusing a proposal, and so on). Instead, the content language of the message is not fixed by the FIPA-ACL specification, and may be chosen by the MAS developer. We express it in XML according to a set of XML-Schemas that we have defined. The messages within AgentSketch have various purposes, such as communicating the availability of new strokes, requesting/providing feedbacks, requesting/providing the sketch interpretation, and so on. Jade offers the means for sending and receiving messages, also to and from remote computers, in a way that is transparent to the agent developer: for each agent, it maintains a private queue of incoming ACL messages that our agents access in a blocking mode. Jade also allows the MAS developer to monitor the exchange of messages using the "sniffer" built-in agent.

A screenshot of the Jade sniffer agent captured during the execution of AgentSketch is shown in Fig. 15. The boxes represent the agents while the arrows the exchanged messages. In particular the first box on the left represents the set of agents included in the jade platform but not monitored by the sniffer agent, while the other boxes (from left to right) represent the Interface Agent, the Input Pre-processing Agent, the SRAs, and the SIA. The figure shows the messages exchanged in the following scenario:

(1) *A new line is drawn*. The Interface Agent sends an inform message containing the stroke attributes to the Input Pre-processing Agent (message 1) and the Input Pre-processing

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

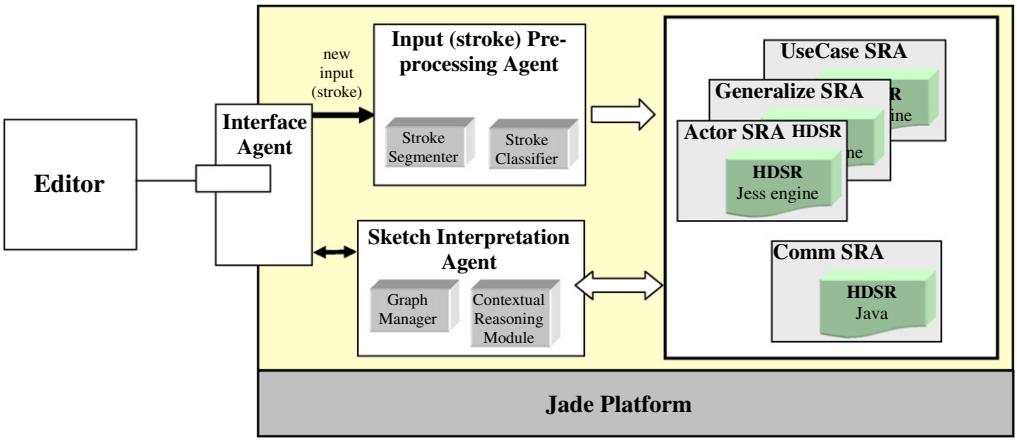


Fig. 14. The AgentSketch architecture.

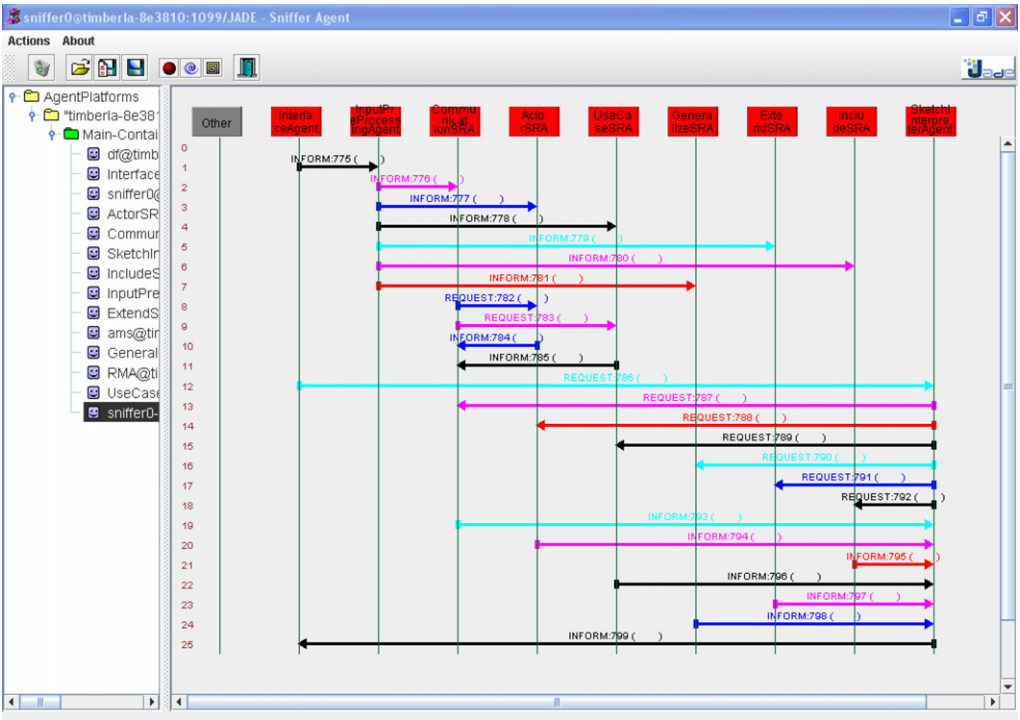


Fig. 15. Agent message exchange in AgentSketch.

Agent sends to all the SRA the new available input in a format that they can understand (messages 2–7). The Communication SRA recognizes, using the new line, a new Communication symbol and sends a feedback request to the Actor SRA and to the Use Case SRA (messages 8 and 9). The Actor SRA and the Use Case SRA check their recognized symbol set and answer to the feedback request (messages 10 and 11).

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

(2) *The interpretation of the sketch is required*. The Interface Agent sends a request message to the Sketch Interpreter Agent (message 12) that forwards it to all the SRA (messages 13–18) in order to collect their recognized symbols. Each SRA sends its set of recognized symbols to the Sketch Interpreter Agent (messages 19–24). When the Sketch Interpreter Agent has collected all the recognized symbols and elaborated a sketch interpretation it sends the sketch interpretation to the Interface Agent (message 25).

The Input pre-processing Agent is composed of two modules, the Stroke Segmenter and the Stroke Classifier. Both modules have been implemented using third parties software and, for their highly modular implementation, might be easily replaced with other pieces of software with the same purpose. The same consideration holds for the Graph Manager that composes the Sketch Interpretation Agent.

The Stroke Segmenter segments strokes into sub-strokes by identifying key points. In this way, symbols can be drawn with multiple pen strokes, and a single pen stroke can contain multiple symbols. Each sub-stroke may be classified by the Stroke Classifier either as a line, an arc, or an ellipse. However, if the user stroke is imprecise, then the Stroke Classifier may classify it as more than one primitive shape (for example, both as a line and as an arc).

In our future work we want to exploit Web Services Technology to implement replaceable, independent, self-consistent HDSRs. This would allow to reach a complete decoupling between agents and HDSR, decoupling which is not fully achieved in our current architecture. In fact, by publishing their functionalities as Web Services, HDSRs might be implemented by anyone and in any language, might be physically stored anywhere, might run on any platform, and might be replaced with a minimal effort. In order to access them, agents should only know their physical address and the specification of the offered services. These services would consist of operations that given a set of stroke classifications, return a recognized symbol, if any.

In our current implementation of AgentSketch, each SRA checks symbol relations for feedback exchange purposes. In particular, exploiting the attributes of symbols recognized by HDSRs, SRAs are able to check a set of relations between symbols (or symbols' portions/points) such as *near*, *intersect*, *touch*, *parallel*, and so on.

The behavior and the structure of the AgentSketch components (both agents and HDSRs) are those described in Sections 4, 5 and 6. In the following, we provide some details on their actual implementation.

Sketch-based user interface. The graphical user interface of AgentSketch has been developed using the Java Swing components and Satin, a toolkit designed to support the creation of 2D pen-based applications [24]. Satin provides a Java Swing component (namely, the Satin *sheet*), that can be easily included in any Java based GUI and that represents a canvas where the user can create strokes based on the path drawn by a pen or mouse.

Strokes are represented as a list of (x, y, time) tuples. A Satin interpreter, associated with the sheet, is automatically notified when a stroke is drawn on the sheet and receives a stroke reference that can be used to obtain information about the stroke (i.e., characterizing points, length, height, width, etc.) and/or to modify it (for example, to change its color or to execute beautification operations). We have developed a customized interpreter that converts the available stroke information into a suitable representation for our system.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

The AgentSketch editor is also able to load and save sketches in InkML format [36]. InkML is an XML data format, designed by the W3C working group, suitable to transfer digital ink data between devices and software components, and for storing hand-input traces for handwriting recognition, signature verification or gesture interpretation. By using InkML, AgentSketch is able to load user sketches previously saved and to import sketches realized with other sketch-based editors.

Stroke Segmenter and Stroke Classifier. The purpose of the Stroke Segmenter is to split each stroke into a set of meaningful sub-strokes enabling the user to draw each symbol with a single-stroke or with a set of strokes. Currently, we have used the segmentation algorithm provided by Satin [24]. In the future, we intend to use the dynamic programming algorithm for stroke segmentation proposed in [37], which is able to split a freehand stroke into the optimal number of line segments and elliptical arcs. When a new stroke is received by the Stroke Pre-Processing Agent it is segmented into a set of sub-strokes (possibly one). The classification process performed by the Stroke Classifier is then applied to each sub-stroke.

The Stroke Classifier analyzes each sub stroke received, and classifies it into one or more domain independent primitive shapes with a set of attributes that the SRAs can use to build the input for AgentSketch HDSRs. In the current version of AgentSketch, the Stroke Classifier is based on HHreco [38], a software library providing multi-stroke symbol recognition capabilities written in Java. HHreco employs a statistical approach to sketched symbol recognition that uses the Zernike moments of strokes as features and enables the user to choose between various classification techniques. We have configured HHreco to use a Nearest Neighbor Classifier, which compares each user stroke with every sample in the training set by computing the normalized Euclidean distance. To classify single strokes in three categories (lines, arcs and ellipses) we have created a training set composed of about fifteen samples for each category. The statistical approach is quite suitable for our problem because usually users draw primitive shapes (arc, line, ellipse) in a similar way.

HDSR. The HDSRs included in AgentSketch have been implemented following the approach proposed by LADDER, described in Section 4, and using the Jess engine [31]. When the SRA receives a new stroke classification, it builds a new fact representing the new stroke classification and adds the fact to the working memory of the Jess recognizer. Then the SRA queries the engine to discover if a new domain symbol has been recognized thanks to the new knowledge just made available. In a similar way, when an SRA receives a feedback request, it executes a Jess query on the working memory of the Jess recognizer in order to check if the symbol in the feedback request message is related through some relationship with some of the symbols it has already recognized. Only the HDSR of the *communication* symbol has been implemented with Java, since it simply tests if one or more received stroke interpretations represent a line longer than a given threshold.

Graph manager and reasoning module. The SIA of AgentSketch contains a reasoning module and a graph manager. The first analyzes the graph of the recognized symbols and determines how to interpret the input drawings. Currently, the interpretation is chosen by considering the contextual information on the conflicting symbols as described in the previous section. If this module is not activated the conflicts are solved considering the information on symbol's accuracy only.

The graph manager is based on JUNG [39], a software library that provides a common and extensible language for modeling, analysis, and visualization of data that can be

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

represented as a graph. Moreover, it includes the implementation of many algorithms from graph theory.

7.2. An experimental evaluation

In order to evaluate the effectiveness of the proposed recognition system we have run an experiment in the domain of Use Case Diagrams. We recruited twenty subjects (15 students and 5 teachers of computer science) with basic knowledge of Use Case Diagrams, to which we briefly introduced AgentSketch, 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. Then we gave them two diagrams of real software systems, which they had to draw using AgentSketch. They knew that they were not being timed and that they could not erase strokes during the editing process. The two diagrams were formed by 16 and 23 symbols, respectively, thus we obtained 780 symbols drawn by the subjects. Fig. 16 shows one of the diagrams.

To analyze the importance of the contextual information for the interpretation of sketched symbols, we compared the results obtained by the context-based recognizer, indicated with CR, with those obtained deactivating the SIA contextual reasoning module, indicated with BR (baseline recognizer). As said above, with this approach, the conflicts are solved by considering the shape's accuracy only.

Table 1 contains some statistics on the recognition performances of both systems. In particular, for each domain symbol we reported: the number of instances drawn by the subjects, those correctly recognized by BR and CR with the relative percentages, the percentage of true negatives (TN), the number of false positives (FP), and the percentage of instances not recognized by any HDSR (US).

The results show that the context-based conflict resolution technique implemented by the SIA considerably improves the overall precision in the recognition. On average, the

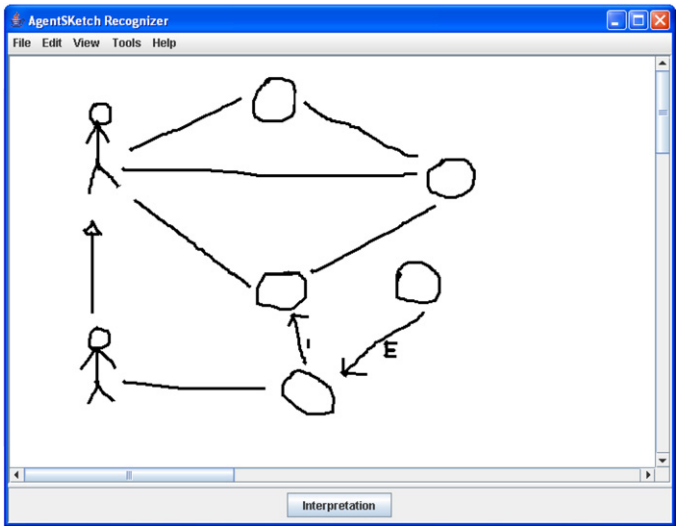


Fig. 16. AgentSketch user interface.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

Table 1
Recognition statistics by symbol

Symbols	#Instances	BR				CR				%US
		#Correct	%Correct	%TN	#FP	#Correct	%Correct	%TN	#FP	
Actor	100	73	73	27	5	84	84	16	1	0
Use Case	260	206	79.23	13.46	15	241	92.69	0	10	7.31
Communication	280	259	92.50	6.07	76	274	97.86	0.71	52	1.43
Include	60	37	61.67	26.66	28	40	66.67	21.66	12	11.67
Extend	40	24	60	30	0	27	67.5	22.5	0	10
Generalize	40	26	65	27.5	4	35	87.5	5	4	7.5
Total	**780**	**625**	**80.13**	**15.13**	**128**	**701**	**89.87**	**5.39**	**79**	**4.74**

% BR = percentage of symbol instances correctly interpreted without using contextual information.
% CR = percentage of symbol instances correctly interpreted using contextual information.
% TN = percentage of real instances of a symbol S recognized as another symbol (true negatives).
# FP = number of symbol instances erroneously recognized as a symbol S (false positives).
% US = percentage of symbol instances unrecognized.

baseline system correctly identified 80% of the symbols while CR correctly identified 90%, with a 39% reduction in the number of recognition errors (i.e., the false positives decrease from 128 to 79). This is particularly true for the *generalize* symbol. This symbol is often recognized as both *generalize* and *communication*, and by considering only the shape accuracy *communication* wins almost all the conflicts. On the other side, the *generalize* symbol is the only one that can link two *actors*. Thus, in many cases the use of contextual information allows to disambiguate the *generalize* symbol properly.

We can also notice that *include* and *extend* are the symbols with worst performances for both BR and CR. This is mainly due to the symbol recognition approach, which is unsuitable to recognize letters.

Table 2 shows the statistics of each diagram for CR only. We can observe that as the number of symbols to be recognized in a diagram increases, the more the number of true negatives and false positives decreases. This is mainly due to the increase of contextual information exchanged between SRAs. On the contrary, the number of unrecognized symbols is dependent from the imprecision in the symbol drawing only.

Table 3 reports the confusion matrix obtained for the CR recognizer. The rows contain the interpretations obtained for the drawn symbols, while the columns indicate the instances (mis-)recognized for a given symbols. As an example, the Generalize row indicates that 35 out of 40 *generalize* symbols drawn are correctly recognized and 2 have been misrecognized as include (true negatives), whereas the Use Case column indicates that 251 *Use Case* symbol instances have been recognized, 10 of which are false positives since they should be part of the *actor* symbol. Thus, a number in the matrix indexed by (row, column) indicate how many times a row symbol is misclassified as a column symbol. The correctly recognized symbols appear on the diagonal.

As shown in the matrix, when an *actor* symbol is misrecognized, more than one symbol interpretation can be produced. This motivates why the first row exceeds the total number of *actor* symbols (100). Indeed, the *actor* head can be interpreted as a *Use Case* and the *actor* body as a set of *communication* symbols, which obtain feedback with the *Use Case*. For the same reason some *include* symbols are misrecognized as *communication* symbols.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

Table 2
Recognition statistics of CR by diagram

	#Symbols	Total no. of symbols	#Correct	%CR	%TN	%FP	%US
Diagram 1	16	320	277	86.56	8.75	11.9	4.69
Diagram 2	23	460	424	92.17	3.04	7.3	4.78

Table 3
Confusion matrix for symbols recognized by CR.

	Symbols	Recognized					
		Actor	Use Case	Communication	Include	Extend	Generalize
Drawn	Actor	84	10	39	4	–	–
	Use Case	–	241	–	–	–	–
	Communication	1	–	274	1	–	–
	Include	–	–	9	40	–	4
	Extend	–	–	4	5	27	–
	Generalize	–	–	–	2	-	35

Finally, analyzing the confusion matrix columns we notice that the *communication* and *include* symbols are those more involved in false positives. This depends on the fact that their shape is contained, or at least in the case of *include*, in all the other symbols except for the *Use Case*. On the contrary, the *extend* symbols are never involved in false positives because the letter E has a little probability to be contained in the other drawing symbols.

8. Related work

In the last two decades several approaches have been proposed for the recognition of freehand drawings. The novelty of our work consists in the exploitation of intelligent agents for the integration and coordination of heterogeneous hand-drawn symbol recognizers.

If we consider the agent 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 [40]. Underneath the QuickSet suite of agents lies a distributed, blackboard-based, multi-agent architecture. The blackboard acts as a repository of shared information and as a facilitator. The agents rely on this facilitator for brokering, message distribution, and notification. The gesture recognition agent recognizes gestures from strokes drawn on the map. Along with the coordinate values, each stroke from the user interface also provides contextual information about objects touched or encircled by the stroke. Recognition results are an *n*-best list of interpretations and an associated probability estimate for each interpretation. This list is then passed to the multimodal integrator that accepts typed feature structures from both the gesture and the parser agents (that interpret natural language sentences), and unifies them. A very similar, but more recent, agent-based multimodal system is Demo, described in [41]. In [42], Achten and Jessurun discuss how graphic unit recognition in drawings can take place using a multi-agent systems approach,

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

where singular agents may specialize in graphic unit-recognition, and multi-agent systems can address problems of ambiguity through negotiation mechanisms. In [43], 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. Once the segmentation stage is completed, an agent-based search of the resultant data begins in an attempt to isolate recognizable high-level features within the sketch. Newly recognized strokes are inserted into an "arena" (a sort of blackboard shared by all the agents, where agents can move), and every single agent is in charge of recognizing a specific high-level feature. Agents are mobile, and they wander around the arena looking for strokes that they can use for recognizing a symbol. The mobile agents gradually become more and more restless over time if they are unsuccessful in their task. The more restless an agent is, the less reputable it is considered by other agents; agents that are happy with their position are indeed considered more reliable, since they are likely to have correctly recognized their symbol, and they exert a major influence on the other agents. Agents can cooperate in an implicit way, if they are close enough, and if the strokes that they are recognizing should be close enough in the sketch. In this case, the close agents become more confident in their recognition. In [21], Juchmes et al. describe EsQUIsE, an interactive tool for free-hand sketches to support early architectural design. The EsQUIsE environment uses pen computer technologies featuring the "virtual blank sheet". Lines drawn on the screen are captured and interpreted in real time thanks to the activity and collaboration of different types of agents. "One stroke" agents recognize single strokes, while "multi-stroke" agents recognize group of strokes with chronological, topological and geometric relations, such as dotted lines. The system also involves agents that recognize individual chars, and a dictionary agent. Each stroke can be marked with a flag by the agents (apart from the dictionary one) that are sure enough to have recognized a symbol by using this stroke. When many different interpretations of the drawing are possible, the agents must work together to propose a pertinent interpretation of the sketch. In particular, agents that put a flag on the same stroke, collaborate to decide which of them is wrong, based on the probability that each of them assigns to the successful recognition. If for example, a long sequence of vertical lines is drawn, the char recognizer agent puts flags on them, since it is almost sure that they are "i"s, and the multi-stroke agent recognizing "hatch marks", also puts flags on them. Since the dictionary agent cannot find any word composed by only "i"s, the text recognizer agent realizes that it was wrong, and removes its flags from the vertical lines. Thus, the sequence of vertical lines is recognized as a hatch mark. In [44], Azar et al. extend the previous multi-agent architecture with the possibility of interpreting also vocal information. The graphical inputs are interpreted by either rule-based agents or model-based agents, while the spoken inputs are interpreted by model-based vocal agents.

When we compare our proposal with those using the agent technology, we find that the main differences lie in the technique exploited for recognizing symbols from stroke classifications, that is established once and for all by the other approaches, and in the intended usage domain of the system, which is very specific for all the implemented systems. In fact, Quickset and Demo have been developed for interacting with maps and for drawing Gantt charts, respectively. The system described in [43] classifies child-like sketches of animals, and EsQUIsE is designed for architectural sketches. The only general-purpose view is provided by Achten and Jessurun. However, their paper is more similar to a feasibility analysis of the adoption of multi-agent techniques to sketch recognition in the

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

very general case, rather than to a proposal of a concrete MAS architecture, like the other papers and ours are.

Regarding the construction of symbol recognizers, besides the ones described in Section 4, many other approaches have been proposed [26,29,30,45,46]. We believe that each of these proposals can be integrated into our framework in spite of the fact that they differ one from another under several aspects, ranging from the identification of the shape of the symbols to the approach used to construct them. For instance, the Rubine recognition engine is a trainable recognizer for single stroke gestures [30]. Gestures are represented by global features and are classified according to a linear function of the features. However, the recognizer is applicable to single-stroke sketches and is sensitive to the drawing direction and orientation. In [26], Apte et al. developed a hard-coded recognizer that examines the geometric properties of the convex hull of a symbol. The recognizer also makes use of special geometric properties of particular shapes. In [29], Kara and Stahovich have developed a symbol recognizer that is capable of learning new definitions from single prototype examples. Moreover, since it is based on a down-sampled bitmap representation, it is particularly useful for drawings with heavy over-stroking and erasing. In [45], Gennari et al. present a circuit diagram recognition system that runs isolated symbol recognizers to generate an interpretation. The symbols are located by considering the areas with high density of pen strokes and the temporal information associated to the segmented input strokes. Then, the candidate symbols are classified using a statistical model constructed on training examples. Their system has as number of strengths such as fast recognition, support for multi-stroke objects and multi-object strokes and arbitrary stroke orderings within each object. However, the approach constrains users to draw non-overlapping symbols and to complete drawing one symbol before beginning the next. Moreover, to reduce the size of the search space, the parsing algorithm sets a limit for the number of segments that a symbol may contain. Finally, in [46] Krishnapuram et al. present a generative probabilistic framework for symbol recognition. Their approach is able to recognize messy drawings using single examples by assuming a Gaussian noise model. However, to keep the search for the optimal segmentation tractable, the authors assume that each subset of stroke fragments considered during segmentation contains no more than seven straight line segments. This might be a severe limitation in domains with moderately complex objects.

Regarding the systems that take into account contextual information for sketch recognition several interesting approaches have been proposed [8,13,47,48]. We believe that the rationale underlying these proposals can be integrated into our SIA reasoning module. Mahoney and Fromherz [47] uses a structural language to model and recognize stick figures. The recognition process starts with the generation of an attributed graph representing the input sketch using the set of lines in the sketch and their relationships. Then, by applying perceptual organization principles such as good continuation and proximity the scene graph is augmented by subgraphs corresponding to possible ambiguities in the interpretation of line relations. Next, the algorithm looks for instances of the model graph in the scene graph by performing subgraph matching. In [13], Kara et al. present a multi-level parsing scheme based on a ''mark-group-recognize'' architecture. The recognition process is guided from ''marker symbols'', which are symbols easy to recognize. The identified marker symbols are used to efficiently cluster the remaining strokes into individual symbol using a trainable symbol recognizer having the advantage of learning new definitions from single prototype examples [29]. The parser uses contextual

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

knowledge to both improve accuracy and reduce recognition times. The applicability of this approach is constrained to the presence of marker symbols in the domain language. In [49], Costagliola et al. present a multi-layer parsing strategy for the on-line recognition of hand-drawn diagrams based on the grammar formalism of SkGs [27]. The recognition system consists of three hierarchically arranged layers that include context-based disambiguation and ink parsing. The recognizer of the diagrammatic language, which is at the top of the hierarchy, applies its context knowledge to prune some of the symbol interpretations, to force the recognition of incomplete symbols, and to select the more suitable interpretation. A similar approach has been developed by Alvarado and Davis [8,10]. They present a parsing approach based on dynamically constructed Bayesian networks that combines bottom-up and top-down recognition algorithm that generates the most likely interpretations first, then actively seeks out parts of those interpretations that are still missing. A major strength of this system is its ability to model low-level errors explicitly and use top-down and bottom-up information together to fix errors that can be avoided using context. Finally, by the observation that in certain domains people draw objects using consistent stroke orderings, Sezgin and Davis have presented a hierarchical recognition model that uses both stroke- and object-level temporal ordering information (i.e., the temporal context) gathered automatically from training data [48]. By exploiting knowledge of how people characteristically follow specific patterns when drawing certain sketches, segmentation and recognition can be perform efficiently and some recognition errors should be avoidable. However, the recognition algorithm requires each object to be completed before the next one is drawn.

9. Conclusions and future work

In this paper we have presented an agent-based framework for interpreting hand-sketched symbols in a context-driven fashion, exploiting heterogeneous techniques for the recognition of each symbol. The recognition process is supported by intelligent agents (SRAs) that manage the activity of hand-drawn symbol recognizers, and coordinate themselves in order to provide efficient and precise interpretations of the sketch to the user. In a certain sense, SRAs can be seen as *mediators* that implement the middle layer for integrating information provided by different data sources (the HDSRs). The approach to information mediation based on intelligent agents has a long tradition [50,51]. In our approach we apply the ideas behind mediation to a new research field.

We have also presented AgentSketch, a sketch recognition system implemented according to the framework exemplified in Section 3. It supports both on-line and off-line recognition mode and can be applied to a variety of domains by providing the suitable recognizers. We have performed a first experimental evaluation of the system on the Use Case Diagrams domain, and the results have indicated that the use of context to disambiguate symbol shapes significantly reduced recognition error over a baseline system that did not consider contextual information.

We are investigating how to make our framework a fully open and dynamic MAS, where new SRAs and new SIAs can be integrated at run-time, and exploring ways to improve the effectiveness of the recognition process, such as the integration of a parser into the SIA for analyzing the syntax of the domain language. This will allow us to improve the accuracy in computing feedback information and to reduce the number of active recognition processes.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

We also intend to investigate user interface issues, such as how to visualize feedback for the recognition results. In the current version of AgentSketch it is the user that decides when to interpret the input drawings, but this can lead to compound errors that can be difficult to detect and correct. On the other hand, recognizing and adjusting shapes as they are drawn can distract the user during the editing process. Thus, sketch-based interfaces should balance between these two modes of showing feedback [9].

Finally, for improving the recognition performances and the usability of the system it would be useful to incorporate information from external knowledge sources in the recognition process. As an example, the user could explicitly indicate the interpretation of a set of strokes interacting with the user interface or giving voice commands. The proposed agent-based framework has the advantage to be easily extensible, so external information can be incorporated in the system independently from the nature of the supplied information.

References

[1] D. Blostein, L. Haken, Using diagram generation software to improve diagram recognition: a case study of music notation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(11) (1999) 1121–1136.
[2] M.D. Gross, The electronic cocktail napkin—A computational environment for working with design diagrams, Design Studies 17 (1) (1996) 53–69.
[3] P. Haddawy, M.N. Dailey, P. Kaewruen, N. Sarakhette, L.H. Hai, Anatomical sketch understanding: recognizing explicit and implicit structure, Artificial Intelligence in Medicine 39 (2) (2007) 165–177.
[4] T. Hammond, R. Davis, Tahuti: a geometrical sketch recognition system for UML class diagrams, in: Proceedings of the AAAI Symposium on Sketch Understanding, AAAI Press, 2002, pp. 59–66.
[5] J.A. Landay, B.A. Myers, Sketching interfaces: toward more human interface design, IEEE Computer 34 (3) (2001) 56–64.
[6] M.W. Newman, J. Lin, J.I. Hong, J.A. Landay, DENIM: an informal web site design tool inspired by observations of practice, Human–Computer Interaction 18 (3) (2003) 259–324.
[7] T.F. Stahovich, R. Davis, H. Shrobe, Generating multiple new designs from a sketch, Artificial Intelligence 104 (1-2) (1998) 211–264.
[8] C. Alvarado, R. Davis, SketchREAD: a multi-domain sketch recognition engine, in: Proceedings of User Interface and Software Technology (UIST'04), Santa Fe, NM, USA, ACM Press, New York, October 24–27, 2004, pp. 23–32.
[9] T. Igarashi, B. Zeleznik, Sketch-based Interaction, IEEE Computer Graphics and Applications 27 (1) (2007) 26–27.
[10] C. Alvarado, R. Davis, Dynamically constructed Bayes nets for multi-domain sketch understanding, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), Edinburgh, Scotland, UK, July 30–August 5, 2005, pp. 1407–1412.
[11] G. Casella, G. Costagliola, V. Deufemia, M. Martelli, V. Mascardi, An agent-based framework for context-driven interpretation of symbols in diagrammatic sketches, in: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06), Brighton, UK, IEEE CS Press, 2006, pp. 73–80.
[12] Object Management Group. UML Specification version 2.0. 〈http://www.omg.org/technology/documents/formal/uml.htm〉, 2005.
[13] L.B. Kara, T.F. Stahovich, Hierarchical parsing and recognition of hand-sketched diagrams, in: Proceedings of User Interface and Software Technology (UIST'04), 2004, ACM Press, 2004, pp. 13–22.
[14] D. Arrivault, N. Richard, P. Bouyer, Fuzzy hierarchical attributed graph approach for handwritten hieroglyphs description, in: Proceedings of 11th International Conference on Computer Analysis of Images and Patterns (CAIP'05), Lecture Notes in Computer Science, vol. 3691, Springer, Berlin, 2005, pp. 748–755.
[15] M. Luck, P. McBurney, O. Shehory, S. Willmott, The AgentLink Community, Agent technology: computing as interaction—a roadmap for agent-based computing, AgentLink III, 2005.
[16] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, The Knowledge Engineering Review 10 (2) (1995) 115–152.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

[17] N.R. Jennings, K.P. Sycara, M. Wooldridge, A roadmap of agent research and development, Journal of Autonomous Agents and Multi-Agent Systems 1 (1) (1998) 7–36.
[18] M. Amer, A. Karmouch, T. Gray, S. Mankovski, An agent model for resolution of feature conflicts in telephony, Journal of Network and Systems Management, 8(3) (2000).
[19] J. Chu-Carroll, S. Carberry, Communication for conflict resolution in multi-agent collaborative planning, in: Proceedings of the International Conference on Multi-Agent Systems (ICMAS'95), 1995, pp. 49–56.
[20] J.L.T. da Silva, V.L. Strube de Lima, Lexical categorical disambiguation using a multi-agent systems architecture, in: Proceedings of the International Conference on Multi-Agent Systems (ICMAS'98), 1998, pp. 417–418.
[21] R. Juchmes, P. Leclercq, S. Azar, A freehand-sketch environment for architectural design supported by a multi-agent system, Computers & Graphics 29 (6) (2005) 905–915.
[22] M. Luck, P. McBurney, C. Preist, A manifesto for agent technology: towards next generation computing, Journal of Autonomous Agents and Multi-Agent Systems 9 (3) (2004) 203–252.
[23] F. Zambonelli, A. Omicini, Challenges research directions in agent-oriented software engineering, Journal of Autonomous Agents and Multi-Agent Systems 9 (3) (2004) 253–283.
[24] J.I. Hong, J.A. Landay, SATIN: a toolkit for informal ink-based applications, in: Proceedings of User Interface and Software Technology (UIST'00), November 5–8, San Diego, California, ACM Press, New York, 2000, pp. 63–72.
[25] T. Hammond, R. Davis, LADDER: A sketching language for user interface developers, Computers & Graphics 29 (4) (2005) 518–532.
[26] A. Apte, V. Vo, T.D. Kimura, Recognizing multistroke geometric shapes: an experimental evaluation, in: Proceedings of User Interface and Software Technology (UIST'93), 1993, ACM Press, New York, pp. 121–128.
[27] G. Costagliola, V. Deufemia, M. Risi, Sketch grammars: a formalism for describing and recognizing diagrammatic sketch languages, in: Proceedings of the Eighth International Conference on Document Analysis and Recognition (ICDAR'05), Seoul, South Korea, IEEE CS Press, 2005, pp. 1226–1230.
[28] M.J. Fonseca, C. Pimentel, J.A. Jorge, CALI—An online scribble recognizer for calligraphic interfaces, in: Proceedings of the AAAI Symposium on Sketch Understanding, AAAI Press, 2002, pp. 51–58.
[29] L.B. Kara, T.F. Stahovich, An image-based trainable symbol recognizer for hand-drawn sketches, Computers & Graphics 29 (4) (2005) 501–517.
[30] D. Rubine, Specifying gestures by example, Computer Graphics 25 (4) (1991) 329–337.
[31] E. Friedman-Hill, Jess, The Java Expert System Shell, Technical Report, Sandia National Laboratories, 2000, ⟨http://herzberg.ca.sandia.gov/jess⟩.
[32] G. Costagliola, V. Deufemia, G. Polese, A framework for modeling and implementing visual notations with applications to software engineering, ACM Transactions on Software Engineering and Methodology, 13(4) (2004) 431–487.
[33] A. Ferreira, M. Vala, J.A. Madeiras Pereira, J.A. Jorge, A. Paiva, A calligraphic interface for managing agents, in: Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'06), 2006, pp. 25–31.
[34] F. Bellifemine, A. Poggi, G. Rimassa. Developing multi-agent systems with JADE, in: Seventh International Workshop Agent Theories Architectures and Languages (ATAL2000), Lecture Notes in Computer Science, vol. 1986, Springer, Berlin, 2000, pp. 89–103.
[35] FIPA ORG, FIPA ACL Message Structure Specification, Document no. SC00061G, 2002.
[36] InkML, ⟨http://www.w3.org/TR/InkML/⟩, October 2006.
[37] Y. Liu, Y. Yu, W. Liu, Online segmentation of freehand stroke by dynamic programming, in: Proceedings of International Conference on Document Analysis and Recognition (ICDAR'05), Seoul, South Korea, IEEE CS Press, 2005, pp. 197–201.
[38] H. Hse, A.R. Newton, Sketched symbol recognition using zernike moments, in: Proceedings of International Conference on Pattern Recognition (ICPR'04), Cambridge, UK, August, IEEE CS Press, 2004, pp. 367–370.
[39] JUNG-the Java Universal Network/Graph Framework, ver. 1.7.5, ⟨http://jung.sourceforge.net/⟩, 2006.
[40] P.R. Cohen, M. Johnston, D. McGee, I. Smith, J. Pittman, L. Chen, J. Clow, Multimodal interaction for distributed interactive simulation, in: Proceedings of Innovative Applications of Artificial Intelligence Conference (IAAI'97), 1997, AAAI Press, pp. 978–985.
[41] E. Kaiser, D. Demirdjian, A. Gruenstein, X. Li, J. Niekrasz, M. Wesson, S. Kumar, Demo: a multimodal learning interface for sketch, speak and point creation of a schedule chart, in: Proceedings of the Sixth International Conference on Multimodal Interfaces (ICMI'04), State College, Pennsylvania, USA, October 14–15, 2004, pp. 329–330.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002

[42] H.H. Achten, A.J. Jessurun, An agent framework for recognition of graphic units in drawings, in: Proceedings of 20th International Conference on Education and Research in Computer Aided Architectural Design in Europe (eCAADe'02), Warsaw, 2002, pp. 246–253.
[43] G. Mackenzie, N. Alechina, Classifying sketches of animals using an agent-based system, in: Proceedings of 10th International Conference on Computer Analysis of Images and Patterns (CAIP'03), Lecture Notes in Computer Science, vol. 2756, Springer, Berlin, 2003, pp. 521–529.
[44] S. Azar, L. Couvreury, V. Delfosse, B. Jaspartz, C. Boulanger, An agent-based multimodal interface for sketch interpretation, in: Proceedings of International Workshop on Multimedia Signal Processing (MMSP-06), British Columbia, Canada, 2006.
[45] L. Gennari, L.B. Kara, T.F. Stahovich, Combining geometry and domain knowledge to interpret hand-drawn diagrams, Computers & Graphics 29 (4) (2005) 547–562.
[46] B. Krishnapuram, C. Bishop, M. Szummer, Generative Bayesian models for shape recognition, in: Proceedings of 10th International Workshop on Frontiers in Handwriting Recognition (IWFHR-9), Japan, IEEE CS Press, 2004, pp. 20–25.
[47] J.V. Mahoney, M.P.J. Fromherz, Three main concerns in sketch recognition and an approach to addressing them, in: Proceedings of AAAI Spring Symposium on Sketch Understanding, AAAI Press, 2002, pp. 105–112.
[48] T.M. Sezgin, R. Davis, Sketch interpretation using multiscale models of temporal patterns, IEEE Computer Graphics and Applications 27 (1) (2007) 28–37.
[49] G. Costagliola, V. Deufemia, M. Risi, A multi-layer parsing strategy for on-line recognition of hand-drawn diagrams, in: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06), Brighton, UK, IEEE CS Press, 2006, pp. 103–110.
[50] S. Bergamaschi, Extraction of informations from highly heterogeneous sources of textual data, in: Proceedings of First International Workshop on Cooperative Information Agents (CIA'97), Lecture Notes in Computer Science, vol. 1997, Springer, Berlin, pp. 42–63.
[51] V.S. Subrahmanian, S-S. Chen, J.A. Hendler, R. Hull, V. Tannen, Smart mediators and intelligent agents (panel), in: Proceedings of the Fifth International Conference on Information and Knowledge Management (CIKM'96), 1996, p. 343.

Please cite this article as: G. Casella, et al., An agent-based framework for sketched symbol interpretation, Journal of Visual Language and Computing (2007), doi:10.1016/j.jvlc.2007.04.002