# Reasoning about Hand-Drawn Sketches: An Approach Based on Intelligent Software Agents

Giovanni Casella[1], Vincenzo Deufemia[2], Viviana Mascardi[1], Maurizio Martelli[1], and Genoveffa Tortora[2]

[1] Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova
Via Dodecaneso 35, 16146, Genova, Italy
{casella,mascardi,martelli}@disi.unige.it
[2] Dipartimento di Matematica e Informatica – Università di Salerno
Via Ponte don Melillo, 84084 Fisciano (SA), Italy
{deufemia,tortora}@unisa.it

**Abstract.** Sketching is a powerful means to represent objects and reason on them. In this paper we describe an integrated environment, conceived as a multi agent system, that brings together sketch recognition functionalities and decision support facilities. In this environment, intelligent agents are exploited both for managing the process of recognition of the sketched objects, and for supporting users in solving decisional problems. We explain our approach and its potential by means of a running example taken from the domain of building's safety.

## 1 Introduction

Sketches play multiple roles [1]: they serve as an external memory to augment the limitation of human cognitive abilities, act as the medium that users use to communicate, and serve as the triggers that enable reasoning [2]. Humans see in a sketch more than a static arrangement of arbitrary symbols: they always consider the meaning underlying the sketch and its potential transformations. A sketch often represents the solution to a problem that the user has in mind. Of course, not every sketch is an admissible solution to that problem. In fact, the placement of sketched objects must respect a set of constraints that depend on the sketch's domain and semantics.

Recent years' experience suggests that a great improvement to hand-drawn sketch recognizers can be introduced by enhancing their level of "intelligence", i.e., enabling them to show an intelligent behavior to the user and to help him/her to find solutions to a problem [3]. A computer system might help the user by providing both recognition functionalities of hand-drawn symbols and domain specific knowledge for extracting the underlying sketch meaning and for supporting the user to reason about it. Users should interact with the drawing components in a natural and user-friendly way for obtaining real-time intelligent feedback from the system, and the system should propose innovative solutions to solve particular problems. An intelligent sketch system should embed a problem solving process involving specific domain knowledge, and should present feedback in

M. Sebillo, G. Vitiello, and G. Schaefer (Eds.): VISUAL 2008, LNCS 5188, pp. 302–314, 2008.

appropriate form, without distracting the user from the tasks in which he/she is engaged [4].

In this paper we present an approach based on agent technology for integrating domain-specific reasoning facilities with sketch recognition functionalities. The system architecture consists of the "Sketch Recognition" component, devoted to recognizing elements drawn by the user and to resolving interpretation ambiguities that may arise; the "User Reasoning Support" component, that checks that symbols drawn by the user respect a set of constraints, and, upon request, suggests a re-organization of them; and the "Interface" component, that provides the graphical interface between the user and the system.

To show how the system might be used in practice, we discuss its adoption in the building's safety domain. Given a building/room plant and a customizable set of criteria to meet as input, our system might allow the user to draw objects that are relevant for safety aspects (fire extinguishers, tables, chairs, closets, windows, doors, lights), to check if the given safety criteria are met by the chosen placement of the objects, and, if not, to reason about how to meet them by re-positioning the objects, and finally to propose suggestions to the user.

The paper is organized as follows. Section 2 motivates this research and describes the case study that will be considered throughout the paper. Section 3 describes why intelligent software agents represent a suitable approach for sketch recognition and reasoning. Sections 4 and 5 describe the proposed agent-based system and its application to the case study. The related work is discussed in Section 6. Finally, conclusions and further research are discussed in Section 7.

## 2 Motivation and Case Study

In many situations, being able to sketch a diagram using an input device, and having the diagram components recognized in an automatic way by a software application, is a great advantage. In fact, this approach allows to save time and paper, to share the diagram with colleagues which are spread all over the world, to teach how a diagram should be correctly drawn, to archive and retrieve it in an electronic form. These advantages are well-known: the literature discusses many examples of software systems able to recognize hand-drawn sketches in very different domains [3,4,5]. However, there are situations where the user's needs go far beyond the use of a software system just for recognizing symbols in the correct way. For example, consider an employee that needs to check that all the safety requirements imposed by his/her country's law, are met by the building where he/she works.

The employee would be surely happy to use a software application that allows him/her to load the building plant in some format, and draw, upon the plant, tables, closets, fire extinguishers, and all the objects that may change location over time, as well as mark some doors as emergency exits. But he/she would be even happier, if the application would allow him/her

– To understand that a table too close to an emergency exit violates the safety constraints, or that the fire extinguishers located within a room, are either not enough, or not located in the most convenient way;
– To move the symbols representing furniture, lights, fire extinguishers, and so on, in order to find a setting that meets the safety constraints;

and would be able to propose, in a pro-active way, a re-arrangement of the objects that meet the safety constraints, in case of their violation.

A software system like this must integrate capabilities coming from three research domains:

– From the domain of automatic hand-drawn sketch recognition, the system must borrow the ability to recognize hand-drawn symbols, and to detect and resolve conflicts among their interpretation;
– From the domain of geometric modeling, it must borrow the ability to model physical objects and to reason about spatial relationships among them;
– Finally, from the domain of artificial intelligence, it must borrow the ability to act as a "pro-active" and "situation aware" expert system, supporting the user in finding violations of constraints, explaining why and where the constraints are violated, and proposing alternative solutions.

Provided that the rules for recognizing free-hand drawn symbols, for verifying the allowed spatial relations among them, and for reasoning on their semantics (determined by the constraints that they must respect), are customizable, such a system would prove useful in many disparate domains. For example, it might help a chemical engineer in reasoning on chemical reactions: the engineer might sketch a chemical as a molecular graph with atoms for nodes and bonds for edges, and the system would be able to recognize the characters that identify atoms and the lines that represent bonds (sketch recognition ability), check that spatial relations among symbols are met (geometric modeling capability), reason over its semantics (artificial intelligent capability).

The domain that we will consider for showing the potential of the proposed system is that of building's safety already introduced in the beginning of this section. Ensuring that a room or a building respects all the safety criteria concerning accessibility of emergency exits, availability of fire extinguishers, position of emergency lights, and so on, is extremely important for saving human lives in case of fires and other calamities. The safety criteria to meet are stated by law, and change from country to country.

As an example, the map in Fig. 1 shows the plant of a public library. The library is composed by three rooms. The one depicted on the upper part of the figure is the place where books are stored and contains four bookcases and a desk. The one below is the reading room containing two tables and a closet. Finally the small one on the right is the bathroom. The system will recognize all the hand-drawn symbols representing the objects relevant for the safety domain, as detailed in Section 5, and will check that they are correctly placed in space according to physical rules (a table cannot intersect a closet), and will reason about these objects following rules determined by the safety domain (a

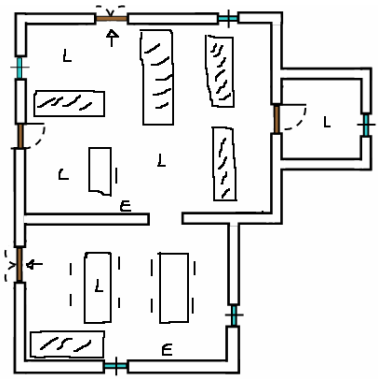

**Fig. 1.** A public library with furniture

closet placed in the middle of a room does not violate any physical or geometric constraint, but could violate a safety constraint if it makes an emergency path longer than a given threshold).

## 3 Intelligent Agents for Recognizing and Reasoning on Hand-Drawn Sketches

The architecture of the system that we propose, consists of the three modules described in detail in Section 4: *User Interface*, *User Reasoning Support*, and *Sketch Recognition*. Recognizing sketched symbols is demanded to the Sketch Recognition module, that integrates algorithms and solutions from the research domain of *automatic hand-drawn sketch recognition*. Verifying that the spatial relationships among symbols are satisfied is demanded to the User Reasoning Support, that must be able to reason about geometric concepts and thus integrates *geometric modeling* capabilities. Finally, reasoning over the drawn symbols in order to support the user in finding their right placement is once again demanded to the User Reasoning Support, that also adds some *artificial intelligence* to the system. Despite to their different abilities, the components that build our system share a common factor: they exploit *intelligent agents* in their implementation.

Following [6,7], an agent can be viewed as a software entity characterized by:

– *Autonomy:* An agent is not passively subject to a global, external flow of control; instead, it has its own internal execution activity, and is pro-actively oriented to the achievement of a specific task.
– *Situatedness:* An agent performs its actions while situated in a particular environment, and it is able to sense and affect such an environment.
– *Sociality:* Agents work in open operational environments hosting the execution of a multiplicity of agents. In these multi-agent systems (MASs), the global behavior derives from the interactions among the constituent agents.

Also *reactivity* is an important feature of agents [8], since they must be able to respond in a timely fashion to changes that take place in the environment. Finally, for many authors an explicit representation of *human-like mental attitudes* such as beliefs, obligations, permissions, is also required for characterizing agents.

According to the definition above, our system is a MAS. In our previous work, we have discussed why most of the entities that compose the Sketch Recognition Module are intelligent agents [9]. The User Reasoning Support module exploits agents as well: we have designed it in such a way that its functionalities are provided by the Decision Support Agent, a "deliberative" agent equipped with both "geometric modeling rules" and "application domain rules" represented by means of Deontic Logic [10] extended with nonmonotonicity. This kind of logic allows the agent to easily model and reason about what the user is permitted to draw, what he/she is forbidden to, what he/she is obliged to, and supports a "default" reasoning thanks to its nonmonotonic component (*human-like attitudes*). The agent will use these rules to check that what the user draws, satisfies both geometric and domain-dependent constraints, and will pursue the goal of avoiding their violation (*pro-activeness*). In case of violation, the agent, without any intervention from the user, will look for an alternative arrangement of the drawn objects (*autonomy*). If an "easy" solution cannot be found in a reasonable amount of time (*reactivity*), the agent will start to interact i with the user in order to collaborate for finding a solution (*sociality*). The virtual sheet where the user draws constitutes the environment of the system, and each agent in the system must perceive and must react to changes that occur inside it (*situatedness*).

In the sequel of the paper, we will show our use of agent technology for providing an intelligent and user-friendly support to users' decisions.

## 4 The Architecture of Our System for Reasoning about Hand-Drawn Sketches

The architecture of the proposed system is shown in Fig. 2. In Section 4.1 we will briefly describe the User Interface and Sketch Recognition modules, whereas in Section 4.2 we describe the Decision Support module by illustrating its functionalities and its architecture. Section 4.3 discusses how all the modules of our system will interact in order to support the user in the most precise and efficient way.

### 4.1 User Interface and Sketch Recognition Modules

The two modules described in this section, already implemented and tested, have been presented in [9].

The User Interface module manages the interaction between the user and the system when the user draws new symbols and when he/she manipulates (deletes, moves, resizes) them. The module integrates a Graphical User Interface for editing sketches. Since different sketch editors may be used for different application domains, new GUIs must be "pluggable" inside the system. In that case, the

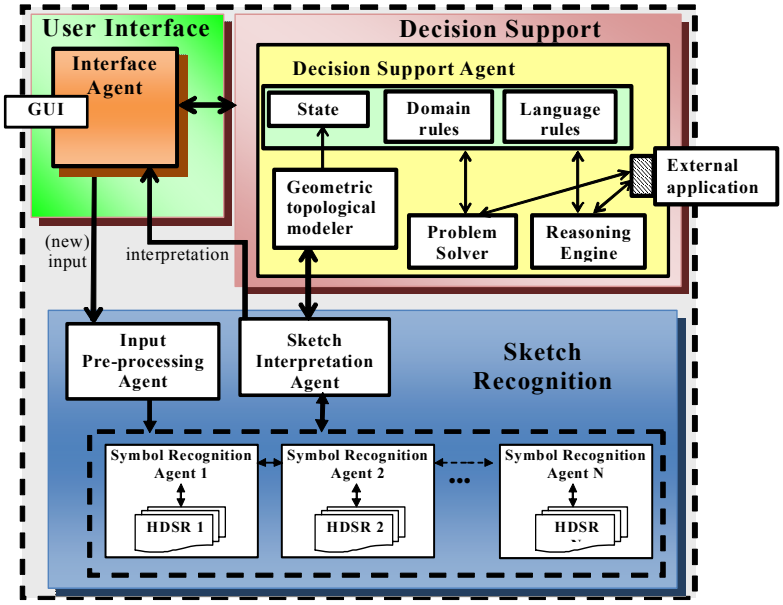


**Fig. 2.** The architecture of the system

Interface Agent is responsible for converting the information produced by the newly plugged editor into a format compliant with our system, and vice versa. It is also responsible for informing the agents belonging to the Sketch Recognition and to the Decision Support modules about both new strokes drawn by the user, and transformations of symbols previously drawn.

The Sketch recognition module provides the functionality of recognizing symbols belonging to a given visual language, hand-drawn by the user using a context based approach. The Input Pre-processing agent is responsible for segmenting and classifying the strokes arriving from the User Interface module. The recognition process performed by the intelligent agents devoted to symbol recognition (Symbol Recognition Agents, SRAs for short) and to the correct interpretation of the sketch (Sketch Interpretation Agent, SIA for short), is based on the knowledge about the language and about the symbols context, which is used for disambiguating the recognized symbols. SRAs exchange contextual information, which is sent to the SIA that solves possible conflicts and gives an interpretation of the sketch drawn. At the lowest level the symbols of the domain language are recognized by applying suitable Hand-Drawn Symbol Recognizers (HDSRs, for short) to the input strokes.

### 4.2 Decision Support Module

**Functionalities:** The Decision Support module, not completely implemented yet, provides the following functionalities.

*Computing both simple and complex geometric and topological relations among symbols.* Given two symbols $S1$ and $S2$, already recognized by the SIA, example of atomic geometric and topological relations between them are "$on\_the\_left\_of(S1, S2)$", "$included\_in(S1, S2)$", "$distance(S1, S2, D)$". The Decision Support Agent (DSA, for short) must be able to exploit its knowledge about geometry and topology in order to decide, given two symbols $S1$ and $S2$, whether $S1$ *intersects* $S2$, or if it is *in front of* it, which is the *distance between* them, and so on. The "geometric and topological modeler" component depicted in Fig. 2 is responsible of computing simple and general geometric and topological relations among the symbols drawn by the user starting from the information provided by the SIA. Since, according to the application domain, the DSA may need to compute relations that are more complex than those calculated by the geometric and topological modeler, and that cannot be decided a priori and once and for all, it must be able to access external components devoted to running optimized, ad hoc algorithms. If, for example, an application working on hand-drawn sketched graphs requires the ability to compute the shortest path, an external module will be accessed in order to run Dijkstra's algorithm on the input graph.

*Modeling language dependent constraints on symbols.* Spatial constraints change from visual language to visual language; for example, in the representation of an electronic circuit, a wire may graphically intersect another wire, while in a Use Case Diagram, no intersections between symbols are allowed.

In our reference domain, where symbols represent architectonic elements and furniture, no partial intersections of symbols are allowed while inclusion of some symbols might be permitted (the symbol representing a light may be entirely included on a table, since it might be placed above it). We may represent rules about relations of symbols using normative concepts such as permitted, forbidden, and obligatory:

–It is permitted that a symbol representing a light is included in a symbol representing a table.
– For any couple of symbols $S1$ and $S2$, it is forbidden that $S1$ intersects $S2$.
– It is obligatory that a symbol representing a door touches a symbol representing a wall.

*Modeling domain dependent constraints on symbols.* Besides constraints depending on the visual language, there are also constraints that depend on the specific application domain of the language. For example, the criteria for arranging furniture in a room in such a way that the comfort of people is ensured, are different from those for ensuring safety, although the set of available symbols and the language constraints they undergo, are the same. Also the domain dependent constraints may be easily expressed in term of "permitted", "forbidden", and "obligatory".

*Verifying that all constraints are satisfied by the current sketch.* The recognized symbols may or may not satisfy the constraints stated by the rules that the DSA possesses. The DSA is in charge of verifying if a violation takes place by running a "reasoning engine" that checks that all the rules are satisfied in the

current state, consisting of the logical representation of the current placement of symbols.

*Helping the user in finding alternative solutions.* In case of violation of constraints, the DSA must inform the user that a violation has occurred. The DSA must also try to find, in a timely fashion, an alternative arrangement of the symbols drawn by the user, such that constraints are respected. A "problem solver" is in charge of this activity. If the alternative arrangement cannot be found in a reasonable amount of time (which may happen, since finding the right placement of objects considering a set of constraints is computationally expensive), the system must start a collaboration with the user for finding a "guided solution" to the constraint violation problem.

**Architecture:** The components of the Decision Support Agent are the following.

*Geometric & topological modeler.* It receives geometric information about symbols drawn so far by the SIA, creates a logical representation of these symbols (the agent's state) consistent with the logical representation used for constraints, and computes a set of "simple" topological and geometric relations, when needed by the engine or by the problem solver that have to verify which constraints are satisfied.

*State and rules.* The DSA is equipped with two sets of rules (also named "constraints" in the paper): those defining what must, can, and cannot be done with the language symbols (language rules), and those defining what must, can, and cannot be done according to the application domain (domain rules). The rules represent the agent program, and operate over the agent state that, as anticipated, consists of the logical representation of drawn symbols and of their placement.

*Reasoning engine.* A reasoning engine reasons about the rules in order to verify that they are respected by the current state. In Computer Science terms, the engine is an interpreter for the agent's program and state. Since the rules may include both "simple" and "complex" relations, the engine must be able to access both the geometric and topological modeler, for computing the former, and any external application that computes the latter. A wrapper, represented by the dashed box attached to the external component in Fig. 2, must implement the conversion of representations between the logical one used by the reasoning engine, and the one used by the external application. The engine must implement a nonmonotonic, forward reasoning. Nonmonotonicity is necessary to avoid stating in the rules everything that is permitted, and everything that is forbidden. Forward reasoning is necessary because the engine starts reasoning from the data it possesses (the current state) and applies recursively all the rules until all of them succeed, or one of them fails, raising a rule violation.

*Problem solver.* Finally, a problem solver looks for alternative solutions that meet the rules, re-arranging the drawn symbols (namely, finding a state different from the current one) in a way that does not raise conflicts with the agents' program. The problem solver must use the engine in order to verify the correctness of the solutions (new states) that it finds.

**State and rules' representation:** For representing the agent's state, we have chosen first-order logic atoms of type *represents*($S$, $Sym$), where $S$ is an identifier and $Sym$ is one among the symbols of the language; *has_bounding_box*($S, BB$), where $BB$ is a couple of points defining the symbol's bounding box; *starts*($S$, $StartingPoint$); and so on. The relations between symbols may be represented as atoms as well: $on\_the\_left\_of(S1, S2)$, $contained\_in(S1, S2)$, and so on.

For representing the agent's rules, we have chosen *deontic logic*, which is the logic to reason about ideal and actual behavior. From the 1950s, von Wright [10] and others developed deontic logic as a modal logic with operators permission (**P**), obligation (**O**), and prohibition (**F**). Providing details about deontic logic is out of the scope of this paper; for more information about this topic, the reader may refer to [11].

By using deontic logic, we can easily express both language and domain rules such as "For any couple of symbols $S1$ and $S2$, it is forbidden that $S1$ intersects $S2$", which is represented by

$$\forall S1, S2\ \mathbf{F} intersects(S1, S2) \tag{1}$$

or, in an electronic engineering domain, "It is obligatory that a symbol representing a CPU is included in a symbol representing a motherboard", that becomes

$$\forall S1, S2((represents(S1, cpu) \wedge represents(S2, motherboard)) \Rightarrow \mathbf{O}\ included\_in(S1, S2))$$

and so on.

The Decision Support Module can be implemented by exploiting the IMPACT framework [12] that allows the user to define rules that embed deontic operators and inside which calls to external code may appear. However, IMPACT is a commercial product and we are instead aiming at developing a free system. For that reason, we are considering the recently developed RBSLA framework (`http://ibis.in.tum.de/projects/rbsla/index.php`) that implements a rule-based system able to integrate deontic modalities and is freely available under GNU license.

### 4.3 System Behavior

The system behavior is described by the following algorithm:

1. Initially, the DSA state is empty
2. The user draws the symbol $S$
3. S is recognized by the Sketch Recognition module, and is passed to the geometric and topological modeler of the DSA that creates its representation as a logical atom and adds it to the current DSA state
4. The DSA runs the reasoning engine using the language and domain rules as program, on the current state
5. If the current state does not violate the rules, then the user can go on drawing (step 2), else

5.1 The user is informed of the violation, and the problem solver is run, in order to find an alternative solution by moving only the last drawn symbol
5.2 If the solution is found in an amount of time lower that a given threshold, then the user is informed of the found solution, else the user is asked to select a symbol that the problem solver will try to move, going back to step 5.1.

## 5 Checking Safety Criteria in Buildings with Our System

The best way to illustrate how our system might be exploited to cope with real problems is through an example.

Fig. 1 shows a building plant where some furniture symbols have been sketched. We suppose that the user needs only to sketch furniture since building plants (including walls, windows, and doors) are loaded from external files. In order to face the problem of building's safety, the user has to sketch

– Furniture having a height that could occlude the light (i.e., wardrobe, libraries, and so on), represented by rectangles containing dotted lines;
– Furniture that cannot obstruct the light (i.e., tables, desks, and so on), represented by empty rectangles;
– Chairs represented by lines placed near low furniture (see Fig. 1).

Moreover the user may sketch an arrow to specify an emergency exit, an "L" to represent a light, and an "E" to represent an extinguisher. In order to support the user in finding the right placement of lights, furniture, and fire extinguishers, the DSA uses a couple of external modules to compute the emergency paths and the poorly illuminated areas. Some of the security constraints are specified by the following deontic rules:

– It is forbidden to have a symbol representing a fire extinguisher behind a symbol representing a closet
$$\forall S1, S2\ ((represents(S1, fire\text{-}extinguisher) \wedge represents(S2, closet)) \Rightarrow \mathbf{F}\ behind(S1, S2)$$
– It is forbidden to have any symbol intersecting an emergency path
$$\forall P, S\ (emergency\_path(P) \Rightarrow \mathbf{F}\ intersects(S, P))$$[1]

Fig. 3(a) shows the plant of Fig. 1 with emergency paths represented by dashed lines and poorly illuminated areas represented by grey rectangles. When the user draws a new symbol (or moves an existing one) the DSA checks if any rule is violated. As an example, if the user draws the table on the left of the lower room of Fig. 3(b), the emergency paths are updated and the DSA detects a

[1] The value $P$ of the logical variable that appears in "$emergency\_path(P)$" must be computed by an external resource, and must be represented in such a way that the geometric and topological modeler can verify the intersects relation between it and the representation of any drawn symbol.

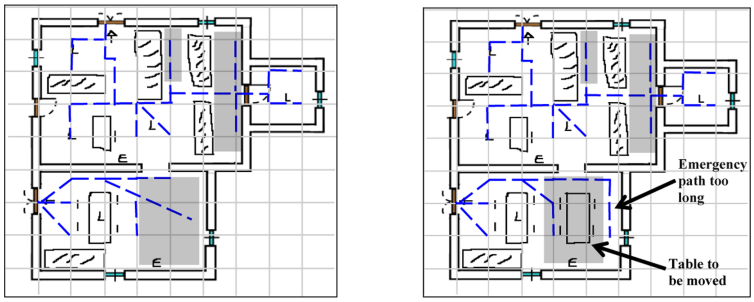


**Fig. 3.** Emergency paths and poorly illuminated areas of a sketched plant (a) and violation of security constraints (b)

constraint violation, i.e., an emergency path becomes too long. By applying the algorithm described in Section 4.3, the DSA suggests to the user to move the table towards the wall at the bottom of the plant, in order to shorten the length of the emergency path.

If the user does not want to move table, he/she can indicate to the system another object to move in order to find a solution. In our example the user could indicate to the system the table placed on the right of the bottom room, and the system would try to find a solution by moving only that table.

## 6 Related Work

In the following we discuss some sketch-based systems providing reasoning functionalities.

If we consider the reasoning functionalities provided by our proposed approach, to our knowledge, only the Design Evaluator system addresses similar issues. Indeed, it is an intelligent sketch system that offers critiquing annotations on drawings to facilitate design reflections [4]. The critiques are generated by applying design rules, coded as Lisp predicates, on the recognized graphical objects. However, "Design Evaluator" presents a monolithic architecture in spite of our modular and easy customizable agent oriented system. Moreover, Deontic Logic enables us to better formalize rules characterizing the considered language/problem.

sKEA (Sketching Knowledge Entry Associate) is designed for capturing knowledge from sketches [13]. sKEA can acquire several kinds of information from the sketches, such as semantic information, positions, what relations one glyph has with others, and which glyphs are conceptually and visually similar. The matching capability of sKEA can be used to suggest users what glyphs would be added and where they would be added. sKEA avoids the recognition problem by requiring the user to indicate when he/she begins and finishes drawing a new object as well as the interpretation of the object.

NuSketch is a sketch system that provides several reasoning services, including analogical reasoning and geographic reasoning [14]. Based on the nuSketch architecture, nusketch Battlespace (nSB), a system specialized for military reasoning, has been created [3]. The system provides a sketching interface for drawing military battle plans, which are understood using qualitative spatial reasoning. However, the system does not attempt to perform shape recognition of the sketches. Rather, it depends on voice input and specific selection procedures from the user to define object types and names.

## 7 Conclusions and Future Works

In this paper we have proposed an integrated environment, conceived as a multi agent system, to support user's reasoning through sketching. We have already demonstrated that the agent technology is suitable for recognizing hand-drawn symbols and for solving ambiguities that may arise in their interpretation [9]. The extension of our system, namely the Decision Support module, also exploits intelligent agents. We have motivated our choice of having an entirely agent-based system, and highlighted its advantages. This choice represents the major difference between our proposal and the related work.

Our future work is to complete the implementation of the system for the presented case study, and to design and develop innovative solutions to enhance user interaction with the system maximizing the advantages of the proposed approach.

## References

1. Purcell, A.T., Gero, J.S.: Drawings and the design process. Design Studies 19, 389–430 (1998)
2. Kavakli, M., Gero, J.S.: Sketching as mental imagery processing. Design Studies 22, 347–364 (1999)
3. Forbus, K., Usher, J., Chapman, V.: Sketching for military courses of action diagrams. In: Proc. of IUI 2003, pp. 61–68. ACM Press, New York (2003)
4. Oh, Y., Do, E.Y.L., Gross, M.: Intelligent critiquing of design sketches. In: Proc. of AAAI Fall Symp. Making Pen-Based Interaction Intelligent and Natural (2004)
5. Landay, J.A., Myers, B.A.: Sketching interfaces: Toward more human interface design. IEEE Computer 34(3), 56–64 (2001)
6. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of ACM 44(4), 35–41 (2001)
7. Lind, J.: Issues in agent-oriented software engineering. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957. Springer, Heidelberg (2001)
8. Jennings, N.R., Sycara, K., Wooldridge, M.: A roadmap of agent research and development. Journal of Autonom. Agents and Multi-Agent Syst. 1, 7–38 (1998)
9. Casella, G., Deufemia, V., Mascardi, V., Costagliola, G., Martelli, M.: An agent-based framework for sketched symbol interpretation. Journal of Visual Languages & Computing 29(2), 225–257 (2008)
10. Von Wright, G.: Deontic logic. Mind, 1–15 (1951)

11. Meyer, J.J.C., Wieringa, R.J.: Deontic Logic in Computer Science. John Wiley and Sons, Chichester (1993)
12. Rogers, T.J., Ross, R., Subrahmanian, V.S.: IMPACT: A system for building agent applications. J. Intell. Inf. Syst. 14(2-3), 95–113 (2000)
13. Forbus, K., Usher, J.: Sketching for knowledge capture: A progress report. In: Proc. of IUI 2002, pp. 71–77. ACM Press, New York (2002)
14. Forbus, K., Ferguson, R., Usher, J.: Towards a computational model of sketching. In: Proc. of IUI 2001, pp. 77–83. ACM Press, New York (2001)