Integrating Agents, Ontologies, and Web Services to Build Flexible Sketch-based Applications

Giovanni Casella[1,2] and Vincenzo Deufemia[2]

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

Abstract. We present an approach based on web services, for building open and dynamic agent societies aimed at hand-drawn sketch recognition. The approach exploits ontologies to enable agents to agree on message semantics and service purposes, standard web services languages to represent agent interaction protocols in a suitable way to be exchanged and handled by agents and web services to expose low-level recognition services. The communication mechanisms that characterize our approach, as well as the modular architecture allow agent societies to self-organize at run time, for gaining the capability of recognizing new domain languages, thus obtaining new flexible sketch-based applications.

1 Introduction

Sketching provides a natural way for humans to design (buildings, software, electronic circuits, and so on), to communicate and cooperate, to share ideas, to transfer information. As an example, sketching allows an architect, or engineer to quickly specify a design. Architects make exploratory sketches before making more definitive schematic design drawings and models, and finally construction and fabrication drawings. Mechanical engineers make sketches as part of a process that also includes calculations, material specifications, and detailed design drawings.

Computers can support users in the sketching process only if they are able to understand the sketch semantics, i.e., they can recognize what the user sketches represent. However, since hand-drawn input tends to be highly variable and inconsistent sketch interpretation turns out to be a difficult task. Sketch recognition systems must robustly cope with the variations and ambiguities inherent in hand drawings, facing the task of grouping a user's pen strokes into clusters representing intended symbols, the task of identifying several symbols in one single stroke, and so on.

In [1] we have exploited intelligent agents to face the diagrammatic sketch recognition problem. The use of agents was inspired by the observation that the

"virtual blank sheet" where the user draws represents a dynamic and unpredictable environment, and the entities devoted to recognize the symbols of some language must be responsive, pro-active, situated, autonomous, and social. In [2] we have presented a Multi-Agent System (MAS) that makes use of collaborating intelligent agents to coordinate a set of heterogeneous symbol recognizers and to generate a sketch interpretation.

This system presents many features suitable for sketch understanding, but it also shows several limitations. In particular, even if the MAS can be built to recognize any domain language, it is impossible to change the domain language while the system is running. This is mainly useful when the users exploit sketches to express new ideas (e.g., an interior architect may wish to add a new shape in its sketched design to represent a new decorative element, such as a flower vase) or when the set of symbols to be recognized can evolve, as for example, in the domain of hieroglyph recognition where new symbols can still be discovered. Another limitation is that if we have a MAS able to recognize a set of symbols S_1, such as use case diagrams, it is not possible to use it to recognize a set of symbols S_2, such as finite state machines, even if S_1 and S_2 share some symbols. Moreover, the MAS does not allow users to integrate interfaces customized to specific tasks. As an example, if the MAS has been designed to facilitate students in the specification of assignment's solutions, it cannot be changed, at run time, into a system that enables teachers to specify solutions to assignments and automatically evaluate student work.

These limitations are mainly due to the lack of flexibility of the agent organization and interactions that, once specified at design time, cannot be adapted to the domain and users' needs at run-time. In this paper, we face these problems by giving to the agents in the MAS suitable means

- to look for the services provided by the other agents,
- to interact with the other agents following heterogeneous interaction protocols, and
- to be able to understand the message semantic and the meaning of the services offered by each agent.

In this way it is possible to recognize new languages and to realize new sketch-based applications changing the interactions between existing agents and/or adding new agents at run-time. In particular, we propose to use ontologies to enable our agents to agree on message semantics and service purposes, and standard web service languages to represent AIP in a way suitable to be exchanged and handled by agents. Web services have also be used to expose low-level recognition services.

The paper is organized as follows. Section 2 introduces the sketch understanding problem and our previously proposed solution. In Section 3 we present the proposed open society for sketch understanding. Section 4 describes how agents publish, reason and learn agent interaction protocols, while Section 5 defines web services for sketch recognition. Finally, Section 6 contains a discussion of related work and conclusions.

2 Sketch Recognition and a MAS to Support it

Sketches are informal drawings created by people to represent abstract concepts and acquired by computers in the format of point chains. The pen trajectory on the screen between each pair of pen-down and pen-up operations, i.e., a unit of user's original sketch input, is named *stroke*. As an example, the human stick figure depicted in Fig. 1(a) is composed of four strokes.

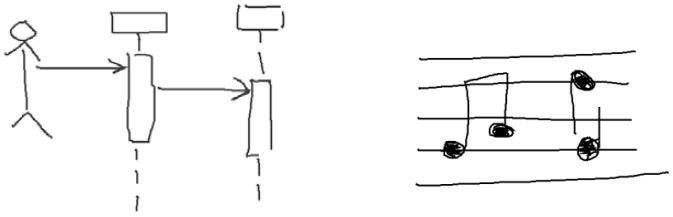

Fig. 1. Two examples of sketches: a sequence diagram (a) and a musical score (b).

Usually, the interpretation of a sketch is performed by classifying the strokes into primitive geometric objects, such as line, arc, and ellipse, and by clustering the primitive shapes into a set of intended symbols. As an example, a human stick is recognized by clustering five line strokes and one ellipse stroke properly related. However, it is worth to note that, according to the user drawing style, the shape of the abstract concepts, namely the symbols, can be drawn using a varying number of strokes (e.g., the two upper rectangles in Fig. 1(a) have been drawn with one stroke and three strokes, respectively), can contain ambiguities (e.g., the recognizer can associate the head of the leftmost note in Fig. 1(b) both to its right and left stems), or be incomplete. These issues make the recognition of sketches a very critical task.

To minimize the recognition mistakes many systems have constrained the user's drawing style (e.g., enforcing users to carefully draw each symbol with a single stroke) making the recognition process easier. However, in order to be really usable and useful in practice, sketch recognition systems should not place constraints on how the users can draw symbols. Indeed, users should be able to draw without having to worry about where to start a stroke, how many strokes to use, in what order to draw the strokes, etc. Beside this, in order to be flexibly adapted to new needs and visual domain languages the recognition system should be able to easily integrate new symbol recognizers without needing to change any other component.

The multi-agent approach to sketch recognition proposed in [2] implements a sketch recognition system having the above features. In particular, it

- manages the variation in drawing style by an ink parsing process that groups and segments the user's strokes into clusters of intended symbols;
- solves the ambiguities due to the possible membership of one stroke to more than one symbol, by analyzing the objects surrounding the ambiguous parts, i.e., the context around them;
- coordinates the behavior of the symbol recognizers in such a way to detect and solve conflicting interpretations of symbols;
- integrates in a seamless way heterogeneous symbol recognizers in order to exploit different techniques for recognizing different symbols.

The recognition approach is based on a MAS composed by intelligent cooperating agents with specific tasks. The MAS contains a set of Symbol Recognizer Agents (SRAs) devoted to recognize the symbols (i.e., their shapes) of a given domain. Each SRA uses an internal recognition algorithm to recognize all the instances of a particular symbol in the sketch, and is able to exchange feedback messages with other SRAs in order to compute contextual information. In particular, when an SRA recognizes a symbol S the belief that S is a correct interpretation increases if other SRAs have recognized symbols that are related with S.

The set of recognized symbols, together with the collected feedbacks, are sent to the Sketch Interpreter Agent (SIA) that analyzes and solves the possible arising conflicts, where a conflict occurs when two or more recognized shapes share one or more strokes.

In the following we exemplify the behavior of the MAS on the UML Use Case Diagram notation. This notation is characterized by the graphical symbols depicted in Fig. 2 and a set of permitted relations between them. Such diagrams are composed of use cases, actors, and connectors among them. In particular, 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. Four types of relationships between use cases are supported by UML: communication, inclusion, extension (visualized as the inclusion but with label $<<extend>>$), and generalization. The only type of relationships that may hold among actors is generalization.

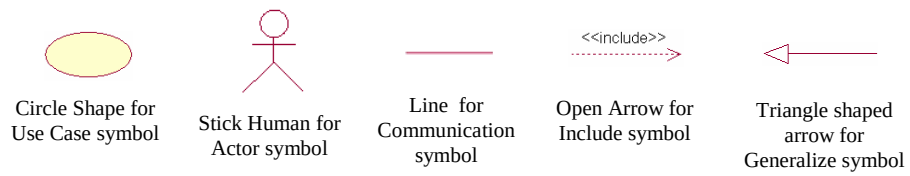

Fig. 2. Use Case Diagram symbols.

Let us suppose that the user draws a Use Case diagram composed by an actor that participates through a communication symbol to a use case, as shown

in Fig. 3. Each SRA in the MAS analyzes the sketch to recognize a particular Use Case symbol. The Actor SRA recognizes actor A_1 (surrounded by a dotted box in the figure). The Communication SRA recognizes C_1 and C_2, where C_1 is part of the actor A_1. Use Case SRA recognizes U_1 and U_2, while Include SRA recognizes I_1. Generalize and Extend SRAs do not recognize any symbol. For each recognized symbol each SRA requests a feedback to the proper SRAs. In particular, the following feedback messages are exchanged:

- Communication SRA obtains a feedback for C_1 from Use Case SRA since it recognized U_1 (and vice versa);
- Include SRA obtains a feedback for I_1 from Use Case SRA since it recognized U_1 (and vice versa);
- Communication SRA obtains a feedback for C_2 from Actor SRA since it recognized A_1 (and vice versa);
- Use Case SRA obtains a feedback for U_2 from Communication SRA since it recognized C_2 (and vice versa).

Finally, each SRA sends the recognized symbols and their feedback to the SIA agent that has to detect and solve conflicts. In particular, A_1, C_1, and U_1 are in conflict, while U_2 and C_2 are considered "unambiguous symbols" because are not in conflict. The unambiguous symbols and the collected feedback are used by the SIA to solve the conflicts. In particular, the SIA applies the following reasoning: A_1 has a feedback from C_2 that is unambiguous, while C_1 and I_1 have a feedback from U_1, but C_1, I_1, and U_1 are in conflict. Then A_1 has been correctly recognized, while C_1 and U_1 have been misrecognized. After the conflict resolution the SIA interprets the sketch as an Actor A_1, a Communication C_2, and a Use Case symbol U_2. The SIA reasoning and the SRAs behavior are detailed in [1].

3 An Open Society for Sketch Understanding

Fig. 4 shows the proposed society of agents and services for sketch understanding, named *AgentSketch*. The society extends the MAS proposed in [1] to support the building of flexible sketch-based applications that can be easily applied across a variety of domains. Indeed, it can be configured to recognize different domain languages and can be extended with new functionalities by adding new agents to the society at run-time.

The features previously described are possible thanks to the proposed architecture, composed by agents and web services, and to the use of suitable ontologies and web service (WS) languages. The latter play a central role to enable agents developed by different organizations and with heterogeneous internal behaviors to interact and to join the society at run-time.

We have identified four groups of agents to build the society:

- **Symbol Recognition Group**: Agents belonging to this group, namely *Symbol Recognizer Agents* (SRAs), are able to recognize a particular domain

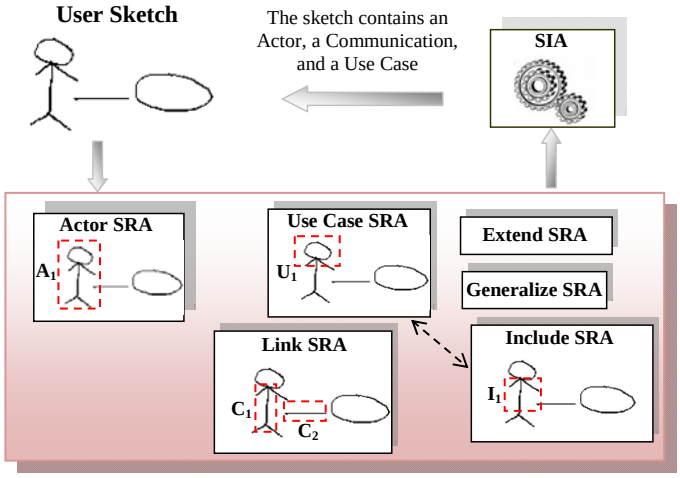

Fig. 3. Use Case diagram interpretation.

symbol and to collaborate with other SRAs in order to obtain contextual information on their recognized symbols. Each SRA interacts with a Shape Recognizer Web Service to recognize a symbol.

- **Sketch Interpretation Group**: Agents that belong to this group, namely *Sketch Interpreter Agents* (SIAs), are able to coordinate the recognition process of a set of SRAs by reasoning on the recognized symbols and elaborating an interpretation of the whole sketch.
- **Domain Expert Group**: A *Domain Expert Agent* (DEA) is able to reason on the sketch interpretations provided by SIAs to face a domain specific task. For example, a DEA could support the user to correctly design circuits by reasoning on the diagrams representing them. Another DEA could be able to reason on Use Case diagrams to help the user to enhance their clarity.
- **Intelligent Interface Group**: *Intelligent Interface Agents* (IIAs) are able to interact with the user in order to enable him/her to draw a diagrammatic sketch and to support him/her in solving a particular task working on the sketch interpretation. An IIA is designed to work on a particular domain language and is customized for a particular purpose. Each IIA collaborates with a SIA to obtain the interpretation of the user sketch and with one or more DEAs to perform some tasks on a previously interpreted sketch.

In order to support agent interactions we have included in our architecture two services, the "*Agent Directory Service*" and the "*Ontology Agent Service*" introduced by the "Abstract Architecture Specification" [3] and by the "Ontology Service Specification" [4], respectively.

An agent uses the Agent Directory Service to register itself and the services that it is able to provide in the "*Agent Directory*". Moreover, an agent queries

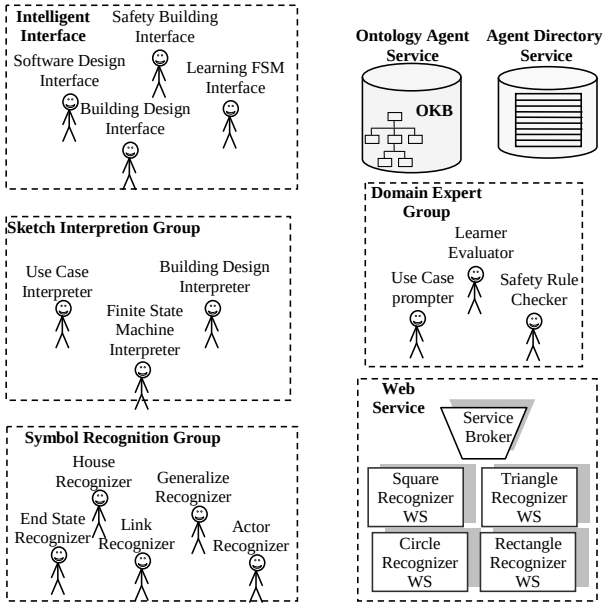

Fig. 4. The AgentSketch society architecture.

the Agent Directory to find other agents able to provide the services that it needs.

We have included in our framework a shared ontological knowledge base, namely AgentSketch OKB, to enable agents to agree on message semantics and service purposes. The Ontology Agent Service supports our community of agents providing services to discover and browse the ontology, and to add instances to the ontology concepts. The OKB will be described in Section 3.1.

Finally, a set of Web Services support SRAs to recognize hand-drawn shapes by offering suitable shape recognition implementations. Agents can find these services through the "Service Broker" also included in our society. Both the Agent Directory and the Service Broker enable agents to discover services, but while the first is suitable to contain agent's information (name, description, interaction protocols) and is used by all the agents, the latter is suitable to contain WSs information (address, WSDL description, and so on) and is used only by SRAs. The WSs of AgentSketch are detailed in Section 5.

The AgentSketch society depicted in Fig. 4 includes WSs able to recognize some basic shapes (i.e., square, arrow, circle, and so on), SRAs able to recognize symbols (i.e., Actor, Generalize, End State, and so on), SIAs able to interpret sketch belonging to different domain languages (i.e., Use Case, Finite State Machine, and so on), IIAs useful for different tasks (i.e., Software Design, Building Design, Learning), and finally, DEA able to furnish domain-specific services (i.e., prompt hints to enhance a Use Case, check if a building design satisfies a set of

security constraints, check if a Use Case diagram is correct for learning purposes). New agents can be added at run-time to increase the society capabilities.

3.1 AgentSketch Ontological Knowledge Base

As stated in [5] to become a member of a society an agent must agree to adhere to the constraints of the system, and in return the agent can benefit from the other members of the society, e.g., their knowledge or services. When an agent enters in the AgentSketch society it must agree to use the AgentSketch OKB to properly communicate with other agents and to understand the services they provide. AgentSketch OKB is shown in Fig. 5.

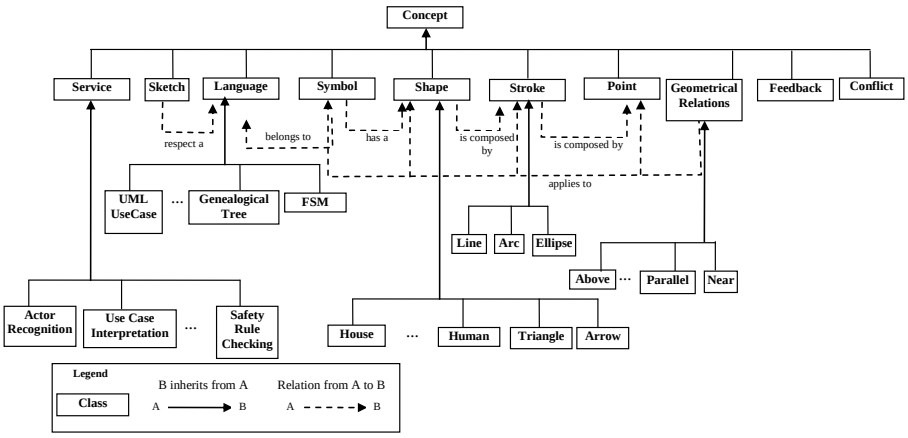

Fig. 5. AgentSketch ontological knowledge base.

Since in AgentSketch society the recognition of a single shape is performed by a single agent, the ontology does not contain low-level concepts about shape recognition, such as shape structure, aggregate of shapes, shape features, and so on. Moreover, the agents do not discuss about their intentions or goals (intentions and goals of each kind of agent are implicitly defined by the services they offer as detailed in the following) so these concepts are not modeled in the ontology.

A common issue is that usually an agent also has an internal ontology used to represent its knowledge and to perform reasoning. In order to use the internal ontology and the shared ontology in a consistent way the agent needs to semantically relate its internal ontology with the shared one. In literature many general approaches for ontology mapping are available [6].

In AgentSketch OKB the class *Concept* acts just as the ontology root: all other classes represent concepts. The class *Service* represents a service offered by an agent. Example of services are: to recognize a particular symbol, to interpret a sketch belonging to a given language, to check if a diagrammatic sketch

satisfies the language constraints, to suggest users a sketch re-arrangement, and so on. The *Sketch* class represents a diagrammatic hand-drawn sketch, i.e., a diagram belonging to a particular language. A *Language* represents a visual domain language and can be described in terms of symbols and rules (i.e., UML Use Case, Genealogy Trees, and so on). In particular, a *Language* is composed of a set of symbols, while the rules define allowed relationships between symbols. A *Symbol* belongs to a particular *Language* and has a particular *Shape*. For example, the Include symbol belongs to the UML use case diagrams and its shape is an Arrow. A *Stroke* represents a user stroke (or a segment of it) and it is composed of a point chain. *Geometrical relations* apply to symbols, shapes, strokes, and points. *Above*, *Under*, *LeftOf*, *Parallel*, *Near* are instances of geometrical relations. The *Feedback* concept represents a feedback sent by an SRA to another about symbol recognition in order to compute contextual information. The *Conflict* concept represents a conflict between two or more recognized symbol interpretations.

3.2 Symbol Recognition and Sketch Interpretation Groups

An SRA is designed to interact with a Shape Recognizer WS for recognizing a particular domain symbol and to collaborate with other SRAs to compute contextual information (feedback exchange). In particular, the WS includes an algorithm to recognize a shape, while the SRA is able to handle the recognized shape as a domain symbol extracting the meaningful features and interacting with other agents to obtain the symbol context. Considering for example the use case diagrams, the SRA devoted to recognize the Include symbol interacts with the WS providing the shape recognition service for the Arrow shape, and collaborates with the SRA devoted to recognize the Use Case symbol.

The role of SIAs is to interpret diagrammatic hand-drawn sketches according to a domain language. The main task of the SIA is to interact with a set of SRAs, to collect their recognized symbols, and to build a coherent sketch interpretation solving the arising conflicts. When a SIA enters in the AgentSketch society it queries the Agent Directory and the OKB to find the SRAs able to recognize the symbols of the domain language. For each SRA it retrieves, from the agent directory, the AIP that it has to follow in order to obtain the service. When the SIA has found the properly SRAs it registers to the Agent Directory and advertises the domain language interpretation service, with the AIP to follow, that it is able to provide. Finally, the SIA browses the OKB to find the language it is able to recognize, if not found then it is added to the instances of the Language concept and the symbols to the instances of the Symbol concept. Using the OKB, the agents can understand the language that the SIA is able to recognize and the symbols that belong to this language.

To obtain the SRA service the SIA has to follows the AIP depicted in Fig. 6 and represented using AUML [5]. The first message sent by the SIA is a request containing: a sketch identifier $sk(id)$, the set of SRAs for the feedback exchange $relatedSRA(sr)$, and the geometrical relations $symb\_rel(r)$ that are allowed between the symbol recognized by the SRA and the shapes recognized by

the related SRAs. The geometrical relations are extracted by the language rules about symbols relationships. As an example, from the rule "a Communication symbol can be connected to an Actor symbol" we extract the relation "the *bounding box* of an actor symbol can be *near* the end or the start point of a Communication symbol". In order to exchange feedback SRA must be able to check these geometric relations starting from the shape recognized by the WS. The SRA can autonomously decide (*outer alternative fragment*) to provide the service ($accept(sk(id))$) or not ($reject(sk(id))$) (e.g., based on the amount of sketches handled at that time by the SRA). If the request is accepted the loop fragment is executed until the protocol ends. In the loop three cases can happen (*inner alternative fragment*):

1. The SIA sends a message, $inform($"$strokes(set)$, $sketch(id)$"), to inform the SRA that the user has drawn a set of strokes that has to be analyzed.
2. The SIA sends a message to the SRA, $request($"$recognized\_shapes$, $sk(id)$"), to request the set of recognized shapes, and the SRA replies sending the shapes with the message $inform($"$recognized\_shapes(set)$, $sk(id)$").
3. The SIA informs the SRA that the user has terminated the drawing by sending the message $inform($"$sketch\_finished$, $sk(id)$").

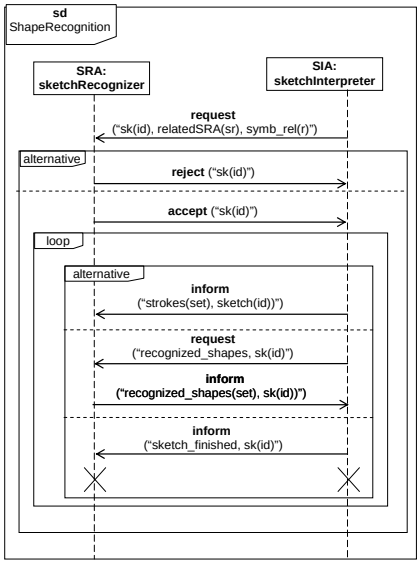

Fig. 6. AUML interaction protocol for the recognition services provided by an SRA.

When a new SRA joins the AgentSketch society it registers to the Agent Directory Service its description (name, address, and so on) and advertises its

recognition service. The SRA adds a standardized textual representation (detailed in Section 4) of the AIP shown in Fig. 6 to its service description. The SRA also adds to the service description the name of the symbol that it is able to recognize. Finally, the SRA browse the AgentSketch OKB and, if it is not present, add the symbol that it is able to recognize to the instances of the *Symbol* concept. Using the OKB another agent can understand the symbol that the SRA is able to recognize.

3.3 Intelligent Interface Agent Group

The main goal of an Intelligent Interface Agent (IIA) is to handle complex interactions with the user in order to enable him/her to draw a diagrammatic sketch and to present him/her feedbacks on the sketch interpretation process (performed by a SIA). Moreover, the IIA can interact with one or more DEA to offer to the user some domain specific services.

To find a SIA able to interpret a given domain language the IIA queries the Agent Directory and the OKB. The SIA retrieves the AIP published by the SIA and follows it to obtain the service. The IIA provides the SIA with the needed information about the sketch (drawn stroke and their attributes, such as spatial coordinates). Moreover, the suitable DEAs are found queering the Agent Directory. An IIA also offers to the users "Intelligent Symbol Manipulations" features. This features support users to easily modify the sketch (for example moving a symbol S while the system automatically re-arrange the symbols related to it) and prevent the violations of the language constraints.

3.4 Domain Expert Agent Group

A DEA is an agent designed to work on the model represented by a diagrammatic sketch (sketch semantic). For example, a DEA could be designed to analyze a diagram representing a building in order to check if some safety rules are satisfied and to prompt some suggestions. The services offered by DEAs are domain-specific and the Agent Interaction Protocols to follow to obtain these services can be very different. However, each DEA registers its provided service and the AIP to obtain it in the Agent Directory.

3.5 An agent society for Use Case Diagram understanding and reasoning

In this section we exemplify the agent society behavior of the intelligent sketch-based application for designing use case diagrams.

Fig. 7 shows the components composing the society and some agent interactions of an application for use case design. In particular, for each shape representing a use case diagram symbol a suitable WS must be available, and for each symbol an SRA must be added to the society. Each SRA searches the suitable WS by querying the Service Broker (arrow 1), and then registers itself in the

Agent Directory (arrow 2). A use case diagram SIA has also to be added to the AgentSketch society. This SIA queries the Agent Directory for finding the SRAs able to recognize the use case diagram symbols and for registering its interpretation service (arrows 3 and 4). Finally, an IIA handles the user interactions by finding the suitable SIA through the Agent Directory (arrow 5). If the user needs advanced domain specific services, one or more DEAs can be added to the society, even at run-time, and the IIA can find them using the Agent Directory (arrow 6).

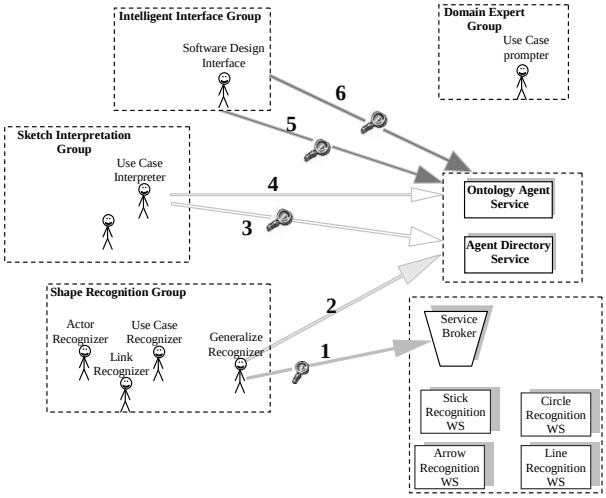

Fig. 7. An agent society for use case diagram recognition-based application.

4 Enabling Agents to Exchange Agent Interaction Protocols

As detailed in the previous sections new agents joining AgentSketch society need to interact with other agents in order to request their services. This can be accomplished only if the agent that need the services follow the AIP associated to them.

Many AOSE methodologies (for example GAIA [7]) take AIP as their starting point to design MASs. Indeed, interacting agents are implemented according to the designed AIPs. AgentSketch society enables agent development at different times and from different development groups by exploiting the advertisement of offered services, and related AIPs to follow in order to obtain these services. In particular, agents can find a service and the AIP to follow by looking into the agent directory. Obviously, the AIPs have to be represented in a standardized unambiguous way, so that the agents can easily handle. A widespread visual

notation used to represent and design AIPs is AUML interaction diagrams [8], an extension of UML sequence diagrams. AUML diagrams are visual diagrams conceived to design MAS by humans and are not suitable for representing AIP in a precise computer processable way. In order to be processed in an automatic way by computers, textual notations are still widely considered to have some significant advantages.

In [9] we have proposed to represent AIPs using a widespread standard textual notation designed for Web Services: the Web Services Business Process Execution Language [10]. WS-BPEL is layered on top of WSDL [11] and provides a language for the formal specification of business protocols describing the mutually visible message exchange of each of the parties involved in the protocol, without revealing their internal behavior. In particular, we have detailed the translation process from AUML to WS-BPEL and we have realized a AUML2WS-BPEL Translator[1], to obtain the automatic translation of an AUML AIP to a WS-BPEL document. In our agent society the AUML2WS-BPEL Translator, available as a set of Java library, can be used by an agent providing a service (i.e., an SRA, a SIA, or a DEA) to generate the WSBPEL representation of the AIP that an agent, that needs the service, namely service consumer has to follow.

The AUML2WS-BPEL Translator can also be used to obtain a Prolog representation of an AIP represented in BPEL. The AIP Prolog representation, as described in [9], can be used to semi-automatically generate the programming code needed by a service consumer agent to execute the AIP. The generated code can be executed in JADE[2] by means of the DCaseLP [12] libraries.

In the AgentSketch society, an agent that needs a service can use the translator to generate the suitable code to handle the AIP published in the Agent Directory by the service provider. For example, an IIA can find a service offered by a DEA in the Directory Agent and can generate the code needed to handle it.

Fig. 8 summarizes the AIP exchange process. The service provider generates the WS-BPEL representation of the AIP associated to a service and stores it in the Agent Directory. The service consumer looks for the service in the Agent Directory and retrieves the BPEL representation of the AIP to follow.

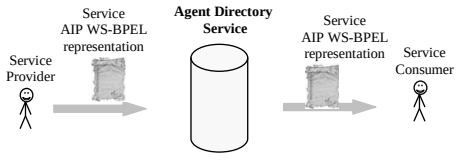

Fig. 8. AIP exchange process.

[1] AUML2WSBPEL Translator, http://www.disi.unige.it/person/MascardiV/Software/AUML2WS-BPEL.html

[2] http://jade.tilab.com/

5 Shape Recognizer Web Services

The implementation of Shape Recognizers (SR) as WSs allows us to integrate in AgentSketch shape recognizers implemented by anyone and in any language, physically stored anywhere, running on any platform, and replaceable with a minimal effort. The use of WSs is also motivated by the low quantity and the simple structure of data to be exchanged between SR and SRA and by the strong availability of standards, design patterns, and development tools. Moreover, the results obtained in the composition of WSs could be exploited to realize complex SRs as composition of simple ones.

Each Shape Recognizer Web Service can internally be based on a different shape recognition approach (for example, Ladder [13] and Sketch Grammars [14] represent suitable choices), however, all WSs must expose the same interface.

The operations that each WS has to expose are described in the following in a "Java like" style:

- **void *start_new_sketch*(Integer *sketch_id*)**
  This operation is used to inform the SR that a new sketch, identified by a number, namely the *sketch_id*, is started. The SR initializes itself to handle the shape recognition associated with the sketch and allocates all the needed resources.
- **void *input_strokes*(Vector *strokes_info*, Integer *sketch_id*)**
  This operation is used to give a set of strokes in input to the SR. The strokes are associated to a sketch identified by the *sketch_id* parameter. Each stroke is represented by an element of the Vector $strokes\_info$ where each element contains the following fields:
  - *Integer stroke_id*: represents the unique *id* in a sketch associated to the stroke
  - *Vector points*: represents the set of key points that characterize the stroke. Each point is represented by its position (x, y coordinates) and by its drawn time t.

  The *input_strokes* operation is called every time the user adds new strokes to the sketch or modify some strokes (for example moving or resizing them). If the SR receives new information about previously known strokes (it can happen if the user moves, resize or modify them), it updates the information associated to it.
- **Vector *input_ns_get_rsymbols*(Vector *strokes_info*, Integer *sketch_id*)**
  This operation is similar to the previous one, but it is used to give a set of strokes in input to the SR and at the same time to ask the set of recognized symbols. The set of recognized symbol is represented by a Vector of $recognized\_symbol\_info$ where each element represents a recognized symbol and it is composed by the following fields:
  - *Integer recognized_symbol_id*: the unique *id* of the symbol;
  - *String recognized_symbol_name*: the recognized symbol name;
  - *Vector stroke_id_vector*: a vector containing all the strokes *id* used to recognize the symbol;

  - *ShapeGeometricAttributes spg*: each shape has associated a set of attributes computed by the SR and defined in the sketch ontology.
- **void *delete_input_strokes*(Vector *strokes_id*, Integer *sketch_id*)**
  This operation is invoked when the user deletes some strokes (*strokes_id*) from the sketch (*sketch_id*).
- **Vector *get_recognized_symbol*(Integer *sketch_id*)**
  This operations is used to ask to the SR the set of recognized symbol for a given sketch *sketch_id*. The output is the same of the *input_ns_get_rsymbols* operation.
- **void *end_sketch*(Integer *sketch_id*)**
  This operation is used to inform the SR that a sketch is finished and all the resource associated with it can be released.

As described in Section 3, WSs advertise agents about their capabilities by means of a Service Broker.

6 Related Work and Conclusions

In the last two decades several approaches have been proposed for the recognition of freehand drawings but few of them exploit agent technology. QuickSet uses a suite of agents for multimodal human-computer communication [15], whereas the approach proposed in [16] uses a system for graphic unit recognition, where singular agents may specialize in graphic unit-recognition, and multi-agent systems can address problems of ambiguity through negotiation mechanisms. EsQUIsE is an interactive tool for free-hand sketches to support early architectural design [17]. The same system has been extended with the possibility of interpreting vocal information [18]. In particular, 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.

Regarding the use of ontologies, Zheng and Sun proposed in [19] a sketch understanding process driven by domain knowledge bases. Their framework allows users to easily adapt the hierarchical understanding process to any domain through the definition of visual concept ontologies.

In this paper we have not concentrated on sketch recognition issues (the suitability and the effectiveness of the agent-based approach for sketch recognition, which is the backbone of the current proposal, have already been discussed in [2]) but we have focused to the flexibility issues of sketch recognition-based system. In particular, we have presented a framework that combines Web Services and Ontologies, for building open and dynamic agent societies aimed at hand-drawn sketch recognition. Ontologies enable agents to agree on message semantics and service purposes, standard web services languages to represent agent interaction protocols in a suitable way to be exchanged and handled by agents, and WSs to expose low-level recognition services. The flexibility of the agent organization and interactions allows us to recognize new languages and to realize new sketch-based applications changing the interactions between existing agents and/or adding new agents at run-time.

References

1. Casella, G., Costagliola, G., Deufemia, V., Martelli, M., Mascardi, V.: An agent-based framework for context-driven interpretation of symbols in diagrammatic sketches. In: Proc. of VL/HCC 06, Brighton, UK, IEEE CS Press (2006) 73–80
2. Casella, G., Deufemia, V., Mascardi, V., Costagliola, G., Martelli, M.: An agent-based framework for sketched symbols intepretation. To appear in Journal of Visual Languages & Computing.
3. Foundation for Intelligent Physical Agents: FIPA abstract architecture specification. http://www.fipa.org/specs/fipa00001/SC00001L.html (2002)
4. Foundation for Intelligent Physical Agents: FIPA ontology service specification. http://www.fipa.org/specs/fipa00086/XC00086D.html (2001)
5. Walton, C.: Agency and the Semantic Web. Oxford University Press (2006)
6. Choi, N., Song, I.Y., Han, H.: A survey on ontology mapping. ACM SIGMOD Record **35**(3) (2006) 34–41
7. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems **3**(3) (2000) 285–312
8. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent UML. In: Proc. of AAMAS'04, IEEE CS Press (2006) 1244–1245
9. Casella, G., Mascardi, V.: Intelligent agents that reason about web services: a logic programming approach. In: Proc. of International Workshop on Applications of Logic Program. in the Semantic Web and Semantic Web Services, Seattle, WA, USA (2006) 55–70
10. Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C.K., Thatte, S., Yendluri, P., Yiu, A., eds.: Web Services Business Process Execution Language (WS-BPEL). Version 2.0. (2005)
11. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C Note. (2001)
12. Gungui, I., Martelli, M., Mascardi, V.: DCaseLP: a prototyping environment for multilingual agent systems. Technical Report DISI-TR-05-20, DISI, Univ. of Genova, Italy (2005)
13. Hammond, T., Davis, R.: LADDER, A sketching language for user interface developers. Computers & Graphics **29**(4) (2005) 518–532
14. Costagliola, G., Deufemia, V., Risi, M.: Sketch Grammars: A formalism for describing and recognizing diagrammatic sketch languages. In: Proc. of ICDAR'05, IEEE Press (2005) 1226–1230
15. Cohen, P.R., Johnston, M., McGee, D., Smith, I., Pittman, J., Chen, L., Clow, J.: Multimodal interaction for distributed interactive simulation. In: Proc. of IAAI'97, AAAI Press (1997) 978–985
16. Achten, H.H., Jessurun, A.J.: An agent framework for recognition of graphic units in drawings. In: Proc. of eCAADe'02, Warsaw (2002) 246–253
17. Juchmes, R., Leclercq, P., Azar, S.: A freehand-sketch environment for architectural design supported by a multi-agent system. Computers & Graphics **29**(6) (2005) 905–915
18. Azar, S., Couvreury, L., Delfosse, V., Jaspartz, B., Boulanger, C.: An agent-based multimodal interface for sketch interpretation. In: Proc. of MMSP-06, British Columbia, Canada (2006)
19. Zheng, W.T., Sun, Z.X.: Knowledge-based hierarchical sketch understanding. In: Proc. of ICMLC'5. (2005) 2838–2843