World Wide Web 2 (1999) 209–218

A Web-based multi-agent system for interpreting medical images

Yi Shang and Hongchi Shi

Department of Computer Engineering and Computer Science, University of Missouri-Columbia, Columbia, MO 65211, USA
E-mail: {shangy,shih}@missouri.edu

A difficult problem in medical image interpretation is that for every image type such as x-ray and every body organ such as heart, there exist specific solutions that do not allow for generalization. Just collecting all the specific solutions will not achieve the vision of a computerized physician. To address this problem, we develop an intelligent agent approach based on the concept of active fusion and agent-oriented programming. The advantage of agent-oriented programming is that it combines the benefits of object-oriented programming and expert system. Following this approach, we develop a Web-based multi-agent system for interpreting medical images. The system is composed of two major types of intelligent agents: radiologist agents and patient representative agents. A radiologist agent decomposes the image interpretation task into smaller subtasks, uses multiple agents to solve the subtasks, and combines the solutions to the subtasks intelligently to solve the image interpretation problem. A patient representative agent takes questions from the user (usually a patient) through a Web-based interface, asks for multiple opinions from radiologist agents in interpreting a given set of images, and then integrates the opinions for the user. In addition, a patient representative agent can answer questions based on the information in a medical information database. To maximize the satisfaction that patients receive, the patient representative agents must be as informative and timely as communicating with a human. With an efficient pseudo-natural language processing, a knowledge base in XML, and user communication through Microsoft Agent, the patient representative agents can answer questions effectively.

1. Introduction

A multi-agent system consists of several interacting agents. In a heterogeneous multi-agent system, agents usually have different roles and work on different tasks. Through communication, collaboration, and coordination mechanisms, they jointly solve a complex problem.

Agent-based computing is a promising paradigm for information processing and software development. Intelligent agents have been studied intensively in areas of computer science and artificial intelligence, and are being applied in a wide variety of applications ranging from small systems such as information filters to large, open, and complex systems such as air traffic control [Bradshaw 1997; Ferber 1999; Jennings and Wooldridge 1998; Nwana and Azarmi 1997; Sycara *et al.* 1996; Weib and Sandip 1996]. In software development of complex systems, divide-and-conquer is a commonly used approach, and modularity and abstraction are powerful methods for reducing complexity. Multi-agent systems represent a general framework for modular design. Complex problems are decomposed into smaller and simpler components that are handled by the corresponding software agents. Agents are specialized in terms of their representation and problem solving techniques and provide a useful abstraction in addressing different aspects of the complex problem. Furthermore, interdependencies between modular system components can be properly managed through cooperation of agents. In this way, a complex problem is solved by a society of cooperating, autonomous problem solvers.

Agent technology is well suited to software development of open and complex computing systems [Ferber 1999; Sycara *et al.* 1996; Weib 1997]. In an open system, the system structure changes dynamically. Its components may be not known in advance, may change over time, and may be implemented by different people using different software tools and techniques. A well-known example of the open system is the Web, which can be viewed as a large, distributed information resource. Through automatic negotiation and cooperation between software agents, multi-agent systems provide a promising approach to harnessing the enormous resources on the Web.

In this paper, we present a Web-based multi-agent system developed for interpreting medical images. One major problem in medical image understanding is that for every image type, such as x-ray or MRI, and for every body organ, such as heart or brain, there exist specific solutions that do not allow for generalization. Just collecting all the specific solutions will not achieve the vision of a computerized physician. The drawback of the traditional approach to medical image understanding is the mixing of the knowledge necessary to solve a given problem with a hard-coded implementation. Object oriented programming allows separation of objects that interact in a certain implementation, but the knowledge is encapsulated in each object. Agent-oriented programming combines the benefits of object-oriented programming and expert system [Shoham 1993].

In our Web-based multi-agent system, there are two major types of intelligent agents: patient representative agents and radiologist agents. The patient representative agent performs two primary tasks: asking opinions about a medical image from radiologist agents and querying a medical information database. For a given medical image, a patient representative agent asks for multiple opinions from radiologist agents that have previously registered with the

agent system facilitator. The integration of the opinions is done by the patient representative agent or by the patient himself. The radiologist agent decomposes the recognition process into separate stages solved by multiple intelligent sub-agents. These sub-agents coordinated by the radiologist agent interact with each other. The interaction between the radiologist agent and its sub-agents is based on the concept of active fusion [Gonzales and Woods 1993; Kopp-Borotschnig and Pinz 1996; Pinz *et al.* 1999; Prantl *et al.* 1996]. The goals of the patient representative agents include (a) understanding natural language, (b) learning on their own by interacting with the radiologist agents, gathering information, and creating their own knowledge base, and (c) responding to patient inquiries in a way that the customer will have little or no idea that they are not really communicating to a real person.

In our preliminary implementation, the radiologist agents have image processing and recognition knowledge, while the patient representative agents have capabilities of pseudo-natural language processing. The database of knowledge is stored in XML, and the communication between the user and the patient representative agents is through Microsoft Agent. The radiologist agents are implemented using the CIAgent agent-programming environment.

This paper is organized as follows. In section 2, we discuss the major components of the multi-agent system for interpretation of medical images. In section 3, we present the design of radiologist agents. In section 4, we present the design of patient representative agents. In section 5, we discuss some implementation issues of the multi-agent system. Finally, in section 6, we conclude the paper and point out some future work.

2. Overview of the multi-agent system

The multi-agent system is composed of two types of agents: patient representative agents and radiologist agents. The patient representative agents take images from the user (usually a patient) for interpretation, and the radiologist agents interpret them. The patient representative agents also answer patient questions based on a medical information database.

Intelligent radiologist agents encapsulate different image analysis algorithms. In addition to the general anatomy knowledge that applies to all imaging modalities, the agents also need to know the specific physics of every modality in order to properly interpret a given image. For example, in MRI images the bone is black and the soft tissue is white, while it is just the opposite in the case of x-ray images. Thus, the agents should be able to incorporate domain knowledge required for different imaging modalities. The radiologist agents are constructed using the image recognition agent model presented in the next section. They have knowledge and some learning capabilities, and may also use exterior algorithms either from libraries or from other agents. The agent system integrates medical image processing knowledge from various sources to make an automatic diagnostic decision or assist a radiologist physician in making a diagnostic decision.

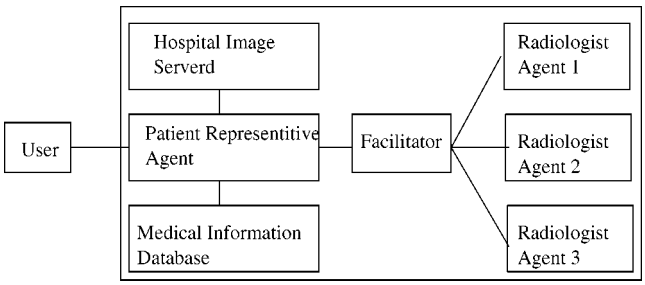


Figure 1. An operation scenario of the multi-agent system for medial image interpretation.

The operation of the multi-agent system is illustrated in figure 1. To make an automated radiological exam in the hospital, the patient first takes the x-ray image and gives it to a patient representative agent. The patient representative agent asks the facilitator to recommend radiologist agents that know how to interpret chest x-ray images. After finding three agents, the patient representative agent passes the patient image to all of them. Every radiologist agent returns the processed version of the image together with its diagnostic decision. The patient representative agent either integrates the answers to make a final diagnostic decision or presents them directly to the patient.

3. Radiologist agents

3.1. Traditional image recognition

Traditional image recognition methods incorporate image specific knowledge into recognition algorithms [Gonzales and Woods 1993]. The result is a program that can only be used for the specific image type that it is created for. The algorithms implemented have reusability to some degree but the knowledge incorporated in the solution cannot be reused or easily changed. Often, the knowledge is incorporated into the algorithms so they cannot be decoupled. To better understand the situation, we examine the traditional image recognition approach shown in figure 2.

The design of a solution with the traditional approach is more an art than a science. In every step, the right algorithm needs to be found and optimized locally. Errors made in an early step such as segmentation must be dealt with in later steps such as recognition. A typical example is chromosome recognition where an over-segmentation produces too many fragments that need to be recombined in order to be recognized, while an under-segmentation might render the given object unrecognizable. Generally speaking, the problems of the traditional approach to image recognition include (a) the knowledge being hard-coded into the algorithms, making every solution unique, (b) every step (e.g., segmentation) being optimized locally without considering the context, and (c) errors made in the earlier steps making the whole problem harder.

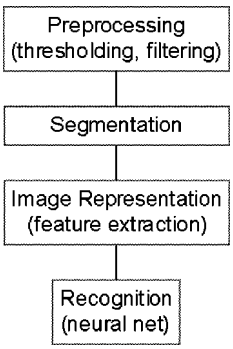


Figure 2. Traditional image recognition approach.

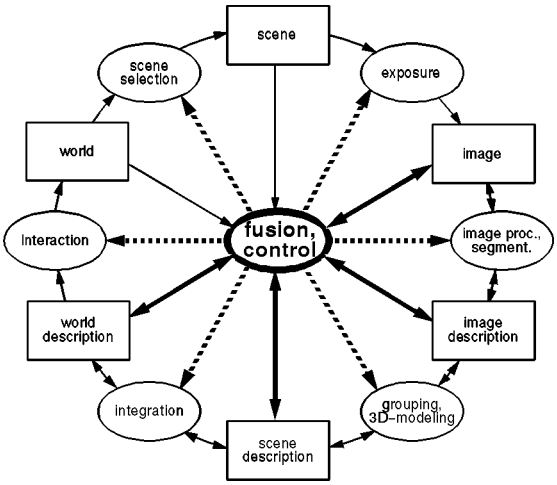


Figure 3. Components of active fusion in image understanding of a 3-D vision system.

3.2. Active fusion approach

To solve some of the problems of the traditional image recognition approach, researchers introduced the concept of active fusion based on information fusion theory [Kopp-Borotschnig and Pinz 1996; Pinz *et al.* 1999; Prantl *et al.* 1996]. Information fusion deals with the integration of information from several different sources, aiming at improving the quality of results. Active fusion extends the paradigm of information fusion. It is not only concerned with the methodology of how to combine information, but also introduces mechanisms in order to select the information sources to be combined.

The active fusion approach has been applied to 3-D vision and to multi-sensor data systems [Prantl *et al.* 1996]. Figure 3 shows the framework of active fusion in image understanding of a 3-D vision system. In this system, the active fusion component constitutes an expert system/control mechanism, which has knowledge about the data sources, the processing requirements, and the effective selection and combination of multi-source data. In the figure, the dotted lines represent data flow and the solid lines represent control flow. The range of actions available to the active fusion component includes selection of viewpoints and activation of image analysis modules (e.g., segmentation and grouping).

3.3. Agent framework based on active fusion

The active fusion approach can be effectively implemented in an agent framework. Figure 4 shows an agent framework that consists of one master image recognition agent performing fusion control and multiple slave agents.

One goal of the framework is to allow the human expert to input the knowledge in the image recognition agent without the need for coding the whole set of algorithms again. The master agent interacts with the environment and controls the action of its sub-agents (slaves). In this framework, the autonomous master and slave agents interact with each other using KQML-like agent communication language. They are able to learn and are persistent. For example, after the segmentation slave agent segments the image, the master (fusion control) agent passes orders to the feature extraction slave agent to label and characterize the objects in the image. If any segmentation problem is suspected, the feature extraction agent can ask the master agent to call the segmentation slave agent in a given context for a given region. When the features for every object are available, the recognition slave agent is invoked. If the master agent suspects that the recognition is not perfect based on its knowledge, it can ask any slave agent to do more work given the new context in which some objects are already recognized, and then it can fuse the new information provided by the slave agent into the known world. Specifically, the constructs of the agents are as follows:

- Master agent
  - task: active fusion controler
  - intelligence: fuzzy rules, decision tree
  - message to slaves: (directive, object, context)
- Segmentation slave agent
  - task: segmentation (morphological, clustering)
  - intelligence: none
  - message to master: (object, context)
- Feature extraction and image recognition slave agents
  - task: feature extraction, image recognition
  - intelligence: fuzzy rules, neural nets
  - message to master: (objects, context)

The master agent sends messages containing the context of the world, the instruction, and the object that the slaves act upon. The slave responds with the same context and the new state of the object (image). The master and slave agents have varying amount of knowledge and learning capabilities.

The new approach to image recognition is based on agent-oriented programming and the concept of active fusion. It combines the benefits of expert system and object-oriented technology [Espeset 1996], allowing separation of

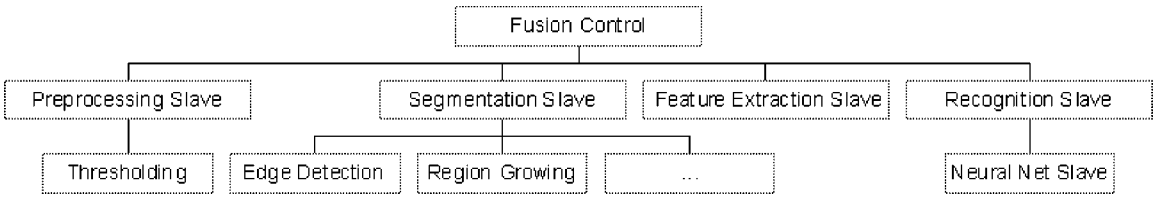


Figure 4. An agent framework for image recognition based on the active fusion concept.

the algorithm from the knowledge, facilitating the global optimization of all the parts of the algorithm, and enabling the recovery from the errors made in earlier steps using knowledge from later steps.

4. Patient representative agent

A patient representative agent sits in the middle of the user and the radiologist agents. It provides a user-friendly, easy-to-use interface to the user based on human-computer interaction techniques. Over the years, many enhancements make on-line intelligent agents more human-like. The first agents handle simple question/answer problems by looking up the question from a large database and supplying the reply by pulling the appropriate field. They severely lack the ability to reply with some type of answer if the question is not in the database. They do not take advantage of the given knowledge to form an answer. Everything has to be hard-coded beforehand by the programmer.

More advanced intelligent agents understand some of the context of a sentence. They reply with information that does not exactly match the question in the database, but has the same meaning and the reply would be valid. This greatly improves the ability of the intelligent agents to come up with an answer. But the agents are still limited to only replying whatever the content in the answer field contained.

The latest intelligent agents start to understand what the question is asking, query the database for the proper answer, and reformulate in a natural language a reply [Jennings and Wooldridge 1998; Microsoft 1999]. The main advantage is that the knowledge base holds searchable information with more generalization over hard-coding the input/output relationship. These intelligent agents can use natural language recognition to understand what the user is asking to create their results. The fact that the answers are no longer hard-coded word for word makes it easy for the intelligent agent to extract the data and present it with different tenses, quantity (plural or singular), or possessiveness.

4.1. Functional components of a patient representative agent

To maximize the satisfaction that patients receive the patient representative agent must be as informative and timely as communicating with a human. With natural language processing, a large knowledge database, and the interaction with the radiologist agents, the patient representative agent can answer questions effectively.

The major functional components of the patient representative agent include communication between the user and the computer, interaction with the radiologist agents, natural language understanding, and knowledge representation and retrieval. Various existing technologies and software packages can be used in the implementation. Thus, the final design usually consists of a hybrid of several different programming languages combined into one entity. Prevalent Internet and Web technologies such as Java and XML are selected in our design because they enable fast prototyping and promote better portability and wider adoption. Particularly, Java and XML are highly complementary technologies of choice in leading, open system solutions. Java provides platform-independence for applications, while XML provides that for data.

4.2. Natural language parser

In natural language understanding, a patient representative agent should understand the user's input and generate appropriate output that will be used to retrieve answers. The natural language parser should run autonomously, without the user's help. Understanding what the user typed in and reducing it into a few key terms is the hardest part. This requires extensive programming and a good knowledge of the English language. Synonyms in the natural language add another problem. People lose interest if they keep hearing the same word repeatedly, but in reality, even with a different spelling a synonym is still the same word. The parser needs to realize this and automatically remove the unessential words from a sentence and break it down into its main parts. The same applies when the agent returns the result. It is a good idea to vary the result ever so slightly for each reply. Working the algorithm backwards, adding more words, and exchanging the reduced small word with more colorful ones will keep the user interested.

The two ends of the spectrum in language parsing are table lookup, which is simple, but usually returns poor results, and natural language understanding, which is very complicated and returns high quality results. We consider a trade-off between quality and complexity in the design, and develop an efficient pseudo-natural language parser that aims to reduce input sentence to minimal content through the following methods:

- split the sentence up into three main parts – subject, verb, and predicate;

- maintain a database of known verbs to split the sentence;
- search the unknown subject and predicate for keywords, assuming the verb is known; and
- return categorized content of the parsed sentence in a usable format.

The implementation of a pseudo-natural language parser benefits from a modification of the natural language parsing. Instead of understanding the meaning of the sentence, the parser reduces the sentence and splits it up into several major components. It is possible to do this effectively, because the knowledge base only contains information about a certain subject. The major components, including the subject, verb, and predicate, are then pattern matched to find the best output. One major disadvantage is that long sentences that do not follow a predefined structure may not be parsed correctly.

4.3. Knowledge representation

In knowledge representation and retrieval, XML is selected due to its flexibility and expressiveness [Bray 1999; Connolly 1999; IBM 1999; xml.com 1999]. The design issues include:

- a special language based on XML is developed for easy search,
- a separate parser is developed to sift through content and locate answer, and
- an XML database provides a more structured access to data over traditional means such as SQL.

In our design, the XML knowledge base contains a structured pseudo-language breakdown of the sentence. The XML parsing function takes in a string and begins to expand the nodes as necessary. First, it starts by expanding which type of sentence the user inputs. Then it expands again based on the verb the user supplies. It does this same type of process repeatedly to narrow down on a solution. By doing this, it allows the program to search the tree faster than searching an entire database.

The knowledge representation should support more advanced queries. A simple implementation uses an "all or nothing" encoded string of XQL to extract the results from the XML database. While this is sufficient to show that it is possible, the power behind it is quite limited. Again, the program needs to search all possibilities of the sentence to ensure an accurate reply. It begins by doing an "AND" on all the variables and slowly, one by one, choosing the least important word each time, add "OR" to the query. Eventually the agent will find the next best result, or it will run out of options and not know what to return. Returning the answer "I don't know" to the user should be avoided, but in reality, we must face that our knowledge base is not ever going to be infinite.

The agent should have the ability to learn from its user and other information sources. One way to generate the knowledge base is to manually code the Q&A relationships. A better alternative is to have the agent ask the user what it should reply if the knowledge is not in the database. This way, a teacher asks the agent questions and whenever it does not know, the teacher provides a solution. The agent can learn in parallel by talking to several teachers at the same time. Another way to train the agent on information is to have it extract the information from the medical database automatically. This is done by creating a second agent that solely reads the content of the database, understands them for the most part, creates Q&A type data, and feeds it to the agent [Sycara *et al.* 1996; Weib and Sandip 1996].

4.4. User interaction

In order to be effective, the agent has to communicate well with the outside world. In our design, Microsoft Agent is used in the communication between the user and the computer. Specially, the consideration is as follows:

- communication to user is done through a separate entity,
- Microsoft Agent allows for voice recognition and other advanced features, and
- communication with Microsoft Agent should be done through JScript or VBScript.

The Java procedure that computes the best answer returns the result back to the HTML page through the JScript function call. The data can then be dumped to the screen as plain text or enhanced with Microsoft Agent [Microsoft 1999]. Consumers generally prefer that the technology be masked behind something they can understand and relate. Microsoft Agents are animated characters that simulate how a person communicates. They create an easy way to retrieve data from the user via keyboard or using the built-in voice recognition software. By using existing technologies such as Microsoft Agent, it cuts out a lot of programming that would normally be required.

5. Implementation issues

There exists a number of agent environments that the multi-agent system can be built upon. Among them, we considered three: the intelligent agent builder (CIAgent) [Bigus and Bigus 1998], the Aglet environment [Lange and Oshima 1998], and a CORBA [Pope 1998] based implemetation of the image analysis modules that are designed using an expert system framework. The Aglet environment has the advantage of supporting mobile agents and open system design. Specialized agents from another Internet "Hospital" can easily get involved in the operation and can come to give a "second opinion." It also allows a comparison of the performance of different agents, something similar to the residency exam. The CORBA model has the same advantages but it generates more network traffic. The implementation of agents in the CIAgent environment is more localized, and the resulting agent system is relatively small, simple, and easy to modify. We

choose CIAgent to implement the radiologist and patient representative agents on the server side. All programs are coded in Java. On the client side, Microsoft Agent is used in the user interface. The implementation of the patient representative agents is broken down into three parts: the pseudo-natural language parser, the XML knowledge base, and the communication between the computer and the user.

5.1. Pseudo-natural language parsing

The underlying algorithm of pseudo-natural language parsing is as follows:

1. Decide which type of sentence the user typed in by looking at punctuation and tell tale signs.
2. Parse the sentence into three distinct sections: subject, verb, and predicate by splitting it on a set of known verbs.
3. Extract the keywords of the subject and the predicate and discard meaningless words.
4. Decide which words are describing words and mark them up separately.
5. Format the new shortened sentence in a tagged way that the next lookup procedure can understand.

The pseudo-natural language parser is written in Java. The user types in input from the keyboard and the input is sent from the HTML page to the Java code via a call from a JScript function. The function uses a set of known words to parse the sentence into its fundamental structure. Since a pseudo-natural language does not need all the words of a sentence it removes all the meaningless words and retains the main keywords of the subject, the predicate, and the type of sentence. Upon processing the string, it posts the results back to the HTML page via another JScript call.

Example.
Where can I quickly locate more information about your growing hospital?

1. The punctuation and the word “Where” at the beginning lets the program know that it is a question and that it is of type “where”.
2. The sentence is then split up into a modified subject, verb, and predicate format:

   Subject: Where can I
   Verb: locate
   Predicate: quickly locate more information about your growing hospital
3. Keywords are extracted and obvious words that show direction are removed. Words like “I” and “You” are known because the conversation is assumed to be between only two people – the computer and the human. “I” is the human and “you” is the computer. A thesaurus is applied and similar words are reduced to one.

   Subject: ~~Where can I~~
   Verb: find
   Predicate: fast ~~locate more~~ information ~~about your~~ growing hospital
4. Decide which words are describing words and mark them up as needed. In this sentence fast is an adverb for locating. Growing is an adjective describing the hospital. These words are set aside for the next step.
5. Markup is done in XML tagged format:

```
<SENTENCE>
  <QUESTION TYPE= “WHERE”>
    <VERB>find
      <ATTRIBUTE>fast</ATTRIBUTE>
    </VERB>
    <PREDICATE>
      <KEYWORD>information</KEYWORD>
      <KEYWORD>hospital
        <ATTRIBUTE>growing</ATTRIBUTE>
      </KEYWORD>
    </PREDICATE>
  </QUESTION>
</SENTENCE>
```

A sample portion of the XML grammar is defined as follows:

```
<?XML version= "1.0"?>
<!DOCTYPE SENTENCE [
<!ELEMENT SENTENCE (QUESTION | STATEMENT |
  EXCLAIMATION | COMMAND)>
<!ELEMENT QUESTION (SUBJECT,VERB,PREDICATE)>
<!ELEMENT SUBJECT (KEYWORD*)>
<!ELEMENT VERB (#PCDATA,ATTRIBUTE*)>
<!ELEMENT PREDICATE (KEYWORD*)>
<!ELEMENT KEYWORD (#PCDATA,ATTRIBUTE*)>
<!ELEMENT ATTRIBUTE (#PCDATA)>
<!ELEMENT STATEMENT (SUBJECT,VERB,
  PREDICATE)>
<!ELEMENT QUESTION (SUBJECT,VERB,PREDICATE)>
<!ELEMENT EXCLAIMATION (SUBJECT,VERB,
  PREDICATE)>
<!ELEMENT COMMAND (SUBJECT,VERB,PREDICATE)>
<!ATTLIST QUESTION #FIXED TYPE (who | what |
  where | when | why | how)>
]>
```

The reason for choosing XML to define the grammar of the string is that parsing a string with no markup is extremely difficult. Without the defined grammar it is hard to tell where the keywords start, how many there are, and to what they belong. With this grammar, the procedure reading in this string to perform the look up knows exactly where all the information is located.

5.2. Knowledge database

There are not many choices to choose from when deciding on a database format. We use either a standard database structure that is queried by SQL, or design a new format specifically suited to this project. We use XML to represent the knowledge in the implementation.

The sample portion of the defined grammar for the previous example can be slightly modified to represent a subset of English:

```
<?XML version= "1.0"?>
<!DOCTYPE ENGLISH [
<!ELEMENT ENGLISH SENTENCE+>
<!ELEMENT SENTENCE (QUESTION | STATEMENT |
  EXCLAIMATION | COMMAND)>
<!ELEMENT QUESTION (SUBJECT,VERB,PREDICATE)>
<!ELEMENT SUBJECT (KEYWORD*)>
<!ELEMENT VERB (#PCDATA,ATTRIBUTE*)>
<!ELEMENT PREDICATE (KEYWORD*)>
<!ELEMENT KEYWORD (#PCDATA,ATTRIBUTE*)>
<!ELEMENT ATTRIBUTE (#PCDATA)>
<!ELEMENT STATEMENT (SUBJECT,VERB,
  PREDICATE)>
<!ELEMENT QUESTION (SUBJECT,VERB,PREDICATE)>
<!ELEMENT EXCLAIMATION (SUBJECT,VERB,
  PREDICATE)>
<!ELEMENT COMMAND (SUBJECT,VERB,PREDICATE)>
<!ATTLIST QUESTION #FIXED TYPE (who | what |
  where | when | why | how)>
]>
```

This grammar is stored in a database as a normal text file. Extracting data is not as easy as it would be in SQL. However, there are similar commands.

XML pattern matching is a valuable way to extract information from a document. For instance, if the XML document contained the information:

```
<ENGLISH>
  <SENTENCE>
    <QUESTION TYPE= “WHERE”>
      <SUBJECT> </SUBJECT>
      <VERB>find</VERB>
      <PREDICATE>
        <KEYWORD>information</KEYWORD>
        <KEYWORD>hospital</KEYWORD>
      </PREDICATE>
    </QUESTION>
    <ANSWER>
      You can find it on the web page of course!
    </ANSWER>
  </SENTENCE>
</ENGLISH>
```

Statements such as this one:

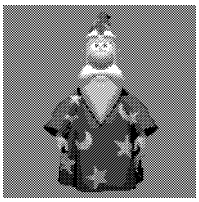
```
//SENTENCE[QUESTION[VERB[text()='find']
  $and$ PREDICATE[KEYWORD[value()=
  'hospital']]
  $and$ PREDICATE[KEYWORD[value()=
  'information']]]]
```

recall the contents of that entry. The entry can then be sent through an XSL style sheet and the corresponding answer can be displayed on the screen or routed through Microsoft Agent.

5.3. Communicating with the user

Writing one's own text-to-speech engine or natural language understanding can be a daunting task. Microsoft Agent does some of the basic tasks for the programmer. The programmer can then use this animated character to talk to the target audience. The code for adding a character to a Web page is as follows:

```
<OBJECT ID= "AgentX" width=0 height=0
    CLASSID= "CLSID:D45FD31B-5C6E-11D1
              -9EC1-00C04FD7081F"
    CODEBASE= "#VERSION=2,0,0,0">
</OBJECT>
```



To make the character speak and move, the simple commands are as follows:

```
function Play() {
    AgentX.Characters.Load("Merlin",
    "Merlin.acs");
    Merlin = AgentX.Characters.Character
    ("Merlin");
    Merlin.Show();
    Merlin.Play("Explain");
    Merlin.Speak("You can find it on the
    web page of course!");
}
```

To make the Web page dynamic, the string where Merlin speaks is replaced by a variable. That variable is the contents of the field “Answer” so when the user posts a question the string is parsed, looked up, and outputted all on the same HTML page.

DHTML is HTML with more advanced features such as changing content on the fly without having to reload the page. It drastically reduces the load on the server, because unlike SQL queries that run on the server the XML queries run on the local client. Users must download the entire XML document to start searching through it. When the document becomes very large, it may be impractical to have each user download the file. In this scenario, it would be better to leave the file on the server.

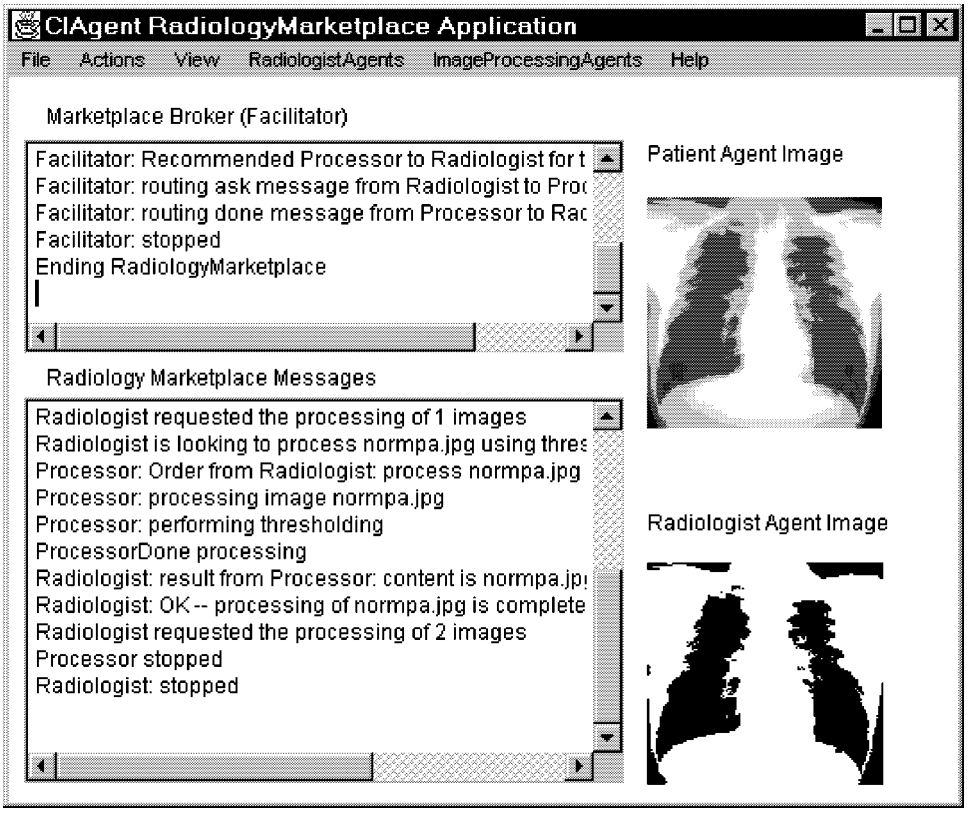


Figure 5. An image presented by the patient representative agent and its processed image from a radiologist agent.

JScript is an easy way to send messages to other components. Instead of running a Java applet full screen in the Web browser, the programmer can use JScript to pass messages back and forth to the Java applet and let DHTML do the redrawing and updating of the screen. JScript also handles the XQL queries and the send/recv information to Microsoft Agent.

5.4. A running example of the multi-agent system

Figures 5 and 6 show a running example of the system, in which the patient representative agent integrates the results returned by three radiologist agents. We implement radiologist agents and slave agents with basic functionalities. The radiologist agents know some basic image processing algorithms. First, the radiologist agent registers with the facilitator as knowing how to process certain images such as x-ray in the example. Assume that the patient representative agent obtains an x-ray image and wants to know where the lungs are in this image. The patient representative agent sends a "recommend-one" directive to the facilitator to find radiologist agents that are able to detect the lungs in the image. As a response to its request, the facilitator sends to the patient representative agent a "tell" directive with the address of each radiologist agent that knows how to do the job. There are three of them in this example. Using this information, the patient representative agent sends the image to each of the radiologist agents together with an "ask" directive. The image is packaged in an "offer" object that contains an image ID, the image file, and the name of the desired algorithm. The radiologist agents process the image using the requested procedure and send back the results using a "done" directive, shown in figure 5.

The patient representative agent uses the answers from the radiologist agents for further integration. After receiving all the response images from the radiologist agents, the patient representative agent uses the responses and a set of rules to find out the answer to the initial question: "Where are the lungs in this image?" The simple rule used in the example is: The region is a lung region if it is indicated so by all the radiologist agents. By going through all the images returned by radiologist agents pixel by pixel, the patient representative agent generates the result in the upper part of figure 6, which is an image with regions most likely to be lungs.

6. Conclusions

One major problem in medical image understanding is that there exist many specific algorithms and implementations that are hard to locate, use, and integrate. Agent-based computing has great potential in addressing this problem by using distributed resources and reducing the programming effort. In this paper, we present a new approach based on agent-oriented programming and the concept of active fusion for medical image understanding. We develop a multi-agent system that consists of patient representative agents

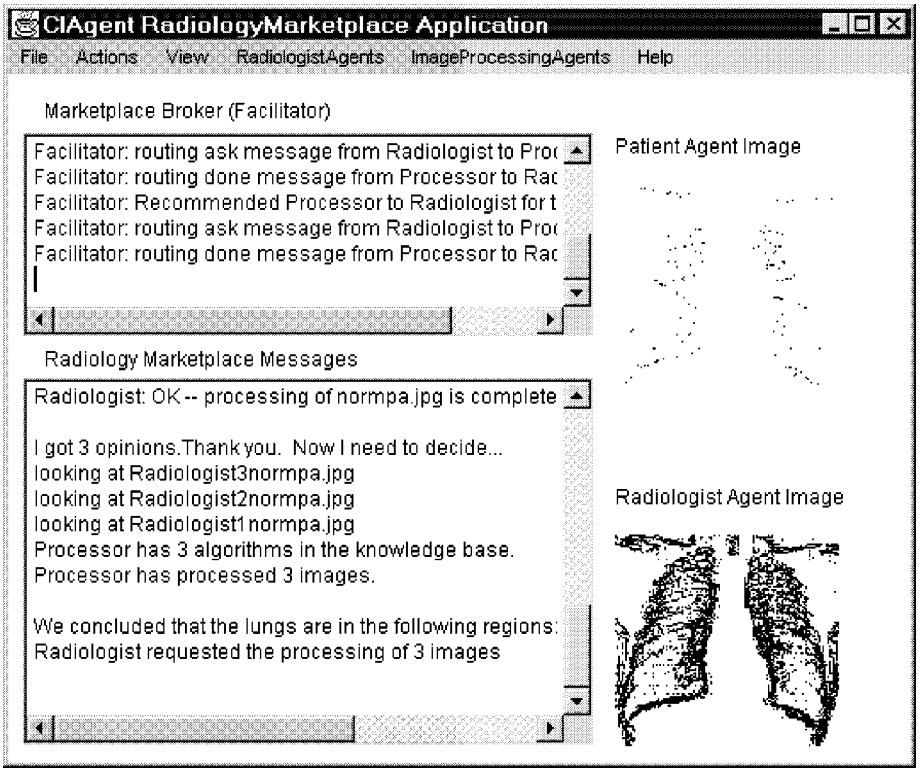


Figure 6. The final answer (the image) determined by the patient representative agent from the images (e.g., the bottom one) returned by all the radiologist agents.

for result integration and radiologist agents for image recognition. The radiologist agents decompose the recognition task based on the concept of active fusion into subtasks for the corresponding slave agents to solve. To maximize the satisfaction that patients receive, the patient representative agents are designed to be as informative and timely as communicating with a human. Our approach of using Java in the initial pseudo-natural language parser, XML as the knowledge base, XQL as the query language and Microsoft Agent to report the findings to the user in an entertaining way appears to be an effective solution.

This work is still quite new and has not been developed to its full capability. Future work includes enhancing the capabilities of agents and the communication, collaboration, and coordination mechanisms in the multi-agent system, improving the pseudo-natural language parser to make it understand the user's input and synonyms better, supporting a more advanced query setup to provide accurate replies in the circumstance of multiple inexact matching, and developing mechanisms for the agents to learn automatically from the users, history profiles, and information on the Web. The ultimate goal is to meet or exceed human capabilities.

Acknowledgements

The authors wish to thank Mr. Mihail Popescu and Mr. Doug Sapp for helping with the implementation of the system.

References

Bigus, J.P. and J. Bigus (1998), *Constructing Intelligent Agents with Java*, Wiley, New York.

Bradshaw, J.M. (1997), *Software Agents*, AAAI Press, The MIT Press, Cambridge, MA.

Bray, T. (1999), "An Introduction to XML Processing with Lark and Larval," `http://www.textuality.com/Lark/.`

Connolly, D. (1999), "Extensible Markup Language Online," `http://www.w3.org/XML.`

Espeset, T. (1996), *Kick Ass Java Programming*, Coriolis Group Books, Scottsdale, AZ.

Ferber, J. (1999), *Multi-Agent Systems*, Addison-Wesley, Reading, MA.

Gonzales, R.C. and R.E. Woods (1993), *Digital Image Processing*, Addison-Wesley, Reading, MA.

IBM (1999), "XML Parser for Java: Another AlphaWorks Technology," `http://www.alphaworks.ibm.com/ formula/XML.`

Jennings, N.R. and M.J. Wooldridge (1998), *Agent Technology: Foundations, Applications, and Markets*, Springer, Berlin.

Kopp-Borotschnig, H. and A. Pinz (1996), "A New Concept for Active Fusion in Image Understanding Applying Fuzzy Set Theory," In *Fuzz-IEEE'96*, New Orleans.

Lange, D.B. and M. Oshima (1998), *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, Reading, MA.

Microsoft (1999), "Microsoft Agents," Microsoft MSDN Online Workshop, `http://www.microsoft.com/msagent.`

Nwana, H.S. and N. Azarmi, Eds. (1997), *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence*, Springer-Verlag, Berlin.

Pinz, A., M. Prantl, H. Gangster, and H. Kopp-Borotschnig (1999), "Active Fusion – A New Method Applied to Remote Sensing Image Interpretation," `http://www.icg.tu-graz.ac.at/CVGroup/Projects/ActiveFusion/activefusion.html.`

Pope, A. (1998), *The CORBA Reference Guide*, Addison-Wesley, Reading, MA.

Prantl, M., H. Gangster, H. Kopp-Borotschnig, D. Siclair, and A. Pinz (1996), "Object Recognition by Active Fusion," In *SPIE Conference on Intelligent Robots and Computer Vision XV*, Vol. 2904, San Jose, CA.

Shoham, Y. (1993), "Agent-Oriented Programming," *Artificial Intelligence 60*.

Sycara, K., K. Decker, A. Pannu, M. Williamson, and D. Zeng (1996), "Distributed Intelligent Agents," *IEEE Expert*.

Weib, G. (1997), "Distributed Artificial Intelligence Meets Machine Learning – Learning in Multi-Agent Environment," In *Lecture Notes in Artificial Intelligence*, Vol. 1221, Springer-Verlag, Berlin.

Weib, G. and S. Sandip (1996), "Adaption and Learning in Multi-agent Systems," In *Lecture Notes in Artificial Intelligence*, Vol. 1042, Springer-Verlag, Berlin.

xml.com (1999), "XML.COM Online," `http://www.xml.com.`