

Intelligent Agents in Medicine

L. Lhotská, *Member, IEEE*, L. Prieto

Abstract— This paper is focused on description of an ongoing project of a multi-agent system application to medical domain. Great interest in multi-agent paradigm is quite natural because a number of software systems have reached such a degree of complexity that it is impossible to control and operate them as monolithic systems. Therefore the effort of decomposition of such systems is a natural procedure. The complex systems should be decomposed into natural functional units that solve partial tasks relatively autonomously and communicate in inevitably minimum range only with the aim to co-ordinate their activities with other units with which they share a global goal. We describe recent results of our ongoing project: the design of a multi-agent system for supporting medical diagnostics in the area of cardiology. Cooperating agents provide a very natural means of automating the pre-processing and (at least partially) the evaluation of a vast amount of medical data utilizing all available medical knowledge. As additional and very important functionality we have decided to design and implement agents that serve for building ontologies from medical resources on the Internet. The current phase of the project is targeted at implementing the system and putting it into full use.

I. INTRODUCTION

THIS paper discusses recent results of the development of a multi-agent system for medical domain. Medicine is characterized by distributed data, information, knowledge, and competence. In addition, all three components (data, information, knowledge) may have different nature: descriptions in natural language, 2D images, measured signals, results of various tests or measurements (usually lists of numbers). They are stored on different media: sheets of paper, photographs, slides, electronic files, books (when considering "classical" knowledge), sometimes personal communications. Usually they are not available in a single place at a particular moment. This distribution represents a major problem when decisions have to be made in a timely fashion.

Modern health care is highly specialized. Complex examination of a single patient involves many expert consultations and laboratory tests. Medical knowledge, examinations and treatment are distributed functionally,

geographically, and also temporally. There is a need for reliable and consistent information flow among all participating subjects with the aim to satisfy the global goal – improved health of a patient. Of course, the necessary information flow is not predictable in extent and structure, but it develops and changes in time due to new knowledge and reactions. To satisfy these requirements and provide adequate decision support, the use of flexible intelligent software support is becoming increasingly desirable. Distributed problem solving and agent technology offer efficient and natural solutions, because they correspond to the main properties of the medical domain, namely distribution of information, problem-solving capabilities, resources, and responsibilities, decision-making with incomplete information, iterative refinement of plans.

Recently there has been growing interest in the application of agent-based systems in health care. The most frequent medical domains in which agents have already been considered are: retrieval of medical knowledge from the Internet [1], decision support systems for monitoring and diagnostic tasks [2] or for home care, distributed patient scheduling within a hospital [3]. AADCare [4] is a system for support of diagnostics, workflow management, and treatment planning. The agent architecture comprises multiple layers of knowledge, a working memory, a communication manager and a human-computer interface. The working memory is similar to a global blackboard [5]. AGIL (Agent-based Information Logistics) [6] is a German research project whose objective is to develop a multi-agent system for optimization of clinical information flows, i.e. for clinical information management. Guardian [2] is a project that was completed in 1996. Its objective was to develop a prototype of intelligent agent system for monitoring intensive-care patients. The system is based on the blackboard control architecture and is composed of heterogeneous software modules. The literature and WWW describe other multi-agent systems and their applications in number of diverse domains - see for example (www.multiagent.com, www.agentlink.org).

II. MULTI-AGENT SYSTEM – CONCEPTUAL DESIGN

We focused on the possibility of utilizing a multi-agent system as a platform for development of an intelligent system for medical diagnostics and monitoring. A knowledge-based model of the agents' mutual awareness (social knowledge) is presented. The tri-base acquaintance (3bA) model formalizing agent's social behaviour

Manuscript received June 30, 2006. This work was supported by the research programme No. MSM 6840770012 "Transdisciplinary Research in the Field of Biomedical Engineering II".

L. Lhotska is with the Czech Technical University in Prague, Gerstner Laboratory, Technická 2, Prague 6, Czech Republic (e-mail: lhotska@fel.cvut.cz).

L. Prieto is with the Universitat Rovira i Virgili, Tarragona, Spain (e-mail: lpr_salamanca@hotmail.com).

knowledge and agent's cooperation neighbourhood knowledge is enhanced to support attention focussing and to ensure a holistic complex perception of the problem [7].

Evaluation of medical data is most often far from straightforward. There is no generally applicable best method or technique to be applied for particular data. Each method has its relative strengths and weaknesses. Some can only produce an approximate solution, but do so comparatively quickly; others are more accurate, but relatively slow. Furthermore, the performance of a given technique is often dependent on the nature of the data set (some work well with noisy data, others do not; some work well with data that has a high signal strength, others work comparatively well with a low signal strength; some can cope with missing data, others do not). The size of the data set may also affect the performance of the technique. When the data set is too large, the user is usually not able to evaluate the quality of data manually, he/she may not be experienced enough, he/she may skip an important part of data, etc. All this is reflected in the basic requirement on the system under consideration: the system needs to be responsive to its problem-solving context.

Considering all the requirements and the nature of the problem domain, the most natural means for modelling and implementing the system is the multi-agent approach.

Each data and signal pre-processing and evaluation technique can be regarded as an autonomous software agent that cooperates, communicates and coordinates, if necessary, with other agents to try to satisfy the global goal. The concept of ADAM (Agents for Diagnostics and Monitoring) [8] defines three basic agent classes, which are necessary to ensure basic functionality and participate in direct problem solving: data collecting agents, which collect data from measuring devices, laboratory test equipment, etc.; data evaluating (interpreting) agents, which pre-process and process collected data using appropriate methods, the results are described in higher-level concepts; and integration agents, which integrate results from processing agents and evaluate them with the aim to reach appropriate conclusions. Based on the evaluation, the integration agent can directly contact data collecting agents and request a new measurement. In addition, the designed system is equipped with complementary agents ensuring system intelligence, namely learning agent, sceptical agent, database agent and info-collecting agent. The database agent collects data about all patients and as a side effect it creates a source for data mining activities. The info-collecting agent searches for complementary information on Internet or among the data collecting agents. The sceptical agent stores past findings relating observations and diagnoses as provided by info-collecting agent or resulting from experience of integrating agents. The learning agent can apply some machine learning methods to data provided by other agents in the community. Most data collection agents have a single functionality; they are not multipurpose machines. All over it, they can take

advantage from the 3bA model since the collection agent has some freedom in the choice of the best integration agent.

Knowledge is divided into four layers, namely domain knowledge (medical knowledge in agent's body), information on co-operating agents (Co-operator Base in agent's wrapper), general problem solving knowledge (Task Base in agent's wrapper), and information on current state of collaborating agents (State Base in agent's wrapper).

In our previous work [9] we suggested a modification of the ADAM architecture for the domain of cardiology. During implementation it proved to be useful to include the info-collecting agent as it was proposed originally. This agent should find relevant information in the Internet resources, extract and analyze it, and finally build ontologies from the retrieved information. During the process of design and implementation it has turned out that a single agent would be too complicated. Therefore the final decision has been to develop a small multi-agent system with required functionality that will communicate with the ADAM system.

III. TOOLS

A. JADE

We were looking for such an environment that would maintain basic multi-agent necessities, which are communication standards, agent administration etc. FIPA (Foundation for Intelligent Physical Agents) [10] provides a set of such standards and reference implementation that have emerged from industrial needs and achievements of the research community. The ADAM system is being built using JADE software framework [11], which is fully implemented in JAVA language. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. The platform independency is important advantage of JAVA solutions. JADE has several interesting features that at least make the process of implementation easier. One of these features is the agent Sniffer that enables user to observe message flow among agents. Another one is the existence of the Agent Management System (AMS). AMS is a mandatory component of the agent platform (AP) and only one AMS exists in a single AP. The AMS is responsible for managing the operation of an AP, such as the creation of agents, the deletion of agents, deciding whether an agent can dynamically register with the AP and overseeing the migration of agents to and from the AP (if agent mobility is supported by the AP).

B. Tools for Information Extraction and Building Ontologies

As a source of reliable and relevant medical information, we have decided to use PubMed. Further we briefly describe

tools that are used for operations with textual information in the developed system.

PubMed [12], available via the NCBI Entrez retrieval system, was developed by the National Center for Biotechnology Information (NCBI) at the National Library of Medicine (NLM), located at the National Institutes of Health (NIH).

Entrez [13] is the text-based search and retrieval system used at NCBI for services including PubMed, Nucleotide and Protein Sequences, Protein Structures, Complete Genomes, Taxonomy, OMIM, and many others. PubMed was designed to provide access to citations from biomedical literature. LinkOut provides access to full-text articles at journal Web sites and other related Web resources. PubMed also provides access and links to the other Entrez molecular biology resources.

HTMLParser [14] is a super-fast real-time parser for real-world HTML. What has attracted most developers to HTMLParser has been its simplicity in design, speed and ability to handle streaming real-world html. The two fundamental use-cases that are handled by the parser are extraction and transformation. It contains filters, visitors, own tags and JavaBeans easy to use. It is a fast, robust and well-tested package.

Lucene [15] from the Apache Jakarta project is a full-featured text search engine library written entirely in Java. It is a high-performance indexing and search engine. "Indexing" means to split sentences into the individual words for storing them in some kind of directory. This directory can then be used for fast lookup of words or combination of words.

Snowball [16] is a language in which stemming algorithms can be exactly defined, and from which fast stemmer programs in ANSI C or Java can be generated. A range of stemmers is presented in parallel algorithmic and Snowball form, including the original Porter stemmer for English. There are two main reasons for creating Snowball:

- The lack of readily available stemming algorithms for languages other than English.
- The lack of promotion of exact implementations of the Porter stemming algorithm.

WordNet [17] is a lexical database for the English language. It groups English nouns, verbs, adjectives and adverbs into sets of synonyms called synsets; it provides short, general definitions, and records various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications.

OWL [18] is an acronym for **Web Ontology Language**, a markup language for publishing and sharing data using ontologies on the Internet. OWL is a vocabulary extension of the Resource Description Framework (RDF) and is derived from the DAML+OIL Web Ontology Language.

Protégé [19] is a free, open-source ontology editor and knowledge-base framework. It is a platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

IV. SYSTEM DESIGN

A. Input and Output

The user can enter some parameters using the graphic interface of the system. These parameters are necessary to build the ontology:

- The main concept from which the ontology will be built. This concept must be in English, since the system has been implemented for this language.
- The highest depth level of the ontology.
- The highest number of articles to analyze for each class.
- The highest number of subclasses to generate for each class.
- The way to decide when a new class will be generated. The options are: the percentage of articles or the number of articles where the class appears.
- The lowest percentage of articles from which a new class is generated.
- The lowest number of articles from which a new class is generated.

The user must enter, at least, the main concept from which the ontology will be built. The other parameters are optional, they already have a default value assigned.

The ontology is built in OWL language from these parameters and it is stored in a file, so that it can be read later on with a tool for reading ontologies like Protégé.

Moreover, the ontology and the articles related to each class are displayed using a graphic user interface at the same time as they have been generated by the system, and the user can visualize them any time. The ontology is displayed in tree form and the user can see the title of the articles related to one of the classes by selecting it in the tree. In addition, if the user clicks on one of the articles, its reference in PubMed will be opened in an Internet browser.

B. MAS Architecture

The designed multi-agent system has two different kinds of agents: Builder agent and Searcher agent

During the process only one builder agent is created. The number of searcher agents depends on the parameters

entered by the user, to be exact, on the number of required searches. So, initially only the builder agent is created, and it has to generate the searcher agents dynamically.

The tasks of the builder agent are:

- Load the graphical user interface to request the parameters to the user and display the result generated in the process (the hierarchy of classes and the articles related to each class).

- Generate the required searcher agents and destroy them when they have finished their work.

- Communicate with each searcher agent to notify the task it has to perform and get the result of this action.

- Analyse the information sent by the searcher agents and discard part of it if necessary.

- Build the ontology at the same time as it receives the required information by the searcher agents until the depth level set by the user is reached or until the process can not continue because there is not information to create new subclasses.

The tasks of each searcher agent are:

- Search in PubMed the articles related to the class that the builder agent has assigned to it.

- Analyse the articles obtained and choose the subclasses that have to be generated from that class:

- Parsing the summary of the articles.

- Searching the previous words of the main word.

- Choosing the nouns and adjectives.

- Checking the number of articles that contain each selected word.

- Choosing the most frequent words.

- Send the information to the builder agent (the new subclasses of the ontology and the articles related to each of them).

C. MAS Operation

When the user enters the main concept from which the ontology will be built (e.g. heart failure), the builder agent generates the first searcher agent so that it perform the search of this first class.

The searcher agent searches in PubMed the class name using the ESearch tool from the Entrez Programming Utilities. This tool returns a web page with a set of Ids, which correspond to the articles related to the class name. The agent parses this web page to extract these IDs using the HTMLParser tool and searches in PubMed the corresponding articles using the EFetch tool from the Entrez Programming Utilities. This tool returns a web page with the information of the articles and the agent parses it to extract the title and summary of each one using the HTMLParser tool.

For each article summary, the searcher agent analyzes the text separating it by words using the StopAnalyzer of the Lucene tool, which already takes charge of ignoring everything what are not letters (e.g. symbols and numbers) and the stop words.

After that, the searcher agent makes several checks with each word of the text, and when a word does not satisfy one of the constraints, it is rejected with no need to continue making the rest of the checks with it. These are the checks:

- If the word is immediately preceding the class name of which the search has been done.

- If it is not too short. We have decided that the minimum length of the selected words is three characters, since putting two we have obtained worse results and putting four we have lost some interesting words of three characters.

- If the word is not in the class name.

- If the class name does not contain any word from the same family (with the same root) that the one which is being analyzed. This check is made with the aid of the Snowball tool.

- If the word is a noun or an adjective. This check is made with the help of the WordNet tool.

- If the class name does not contain any synonym of the word which is being analyzed in its different meanings as noun and its different meanings as adjective. This check is made with the WordNet tool.

When the checks are finished the searcher agent has to choose, among the words that have satisfied all the constraints, the n that have appeared in the articles a greater number of times, being n a parameter entered by the user. Moreover, this parameter is the highest number of subclasses that each class will have.

But there is still the last constraint that these n words have to satisfy: they must have appeared in a minimum number of articles or in a minimum percentage of the number of analyzed articles, according to the criterion that the user has chosen.

Once the searcher agent has the definitive words (e.g. breast and prostate) and therefore, the new subclasses that will be created in our ontology (e.g. breast cancer and prostate cancer), it sends the following information to the builder agent:

- The new subclasses that it has to add to the ontology.

- The title and ID of the articles obtained searching the parent class, but only those that no contains the new subclasses. For example, if it has searched the class 'cancer' and it has obtained the subclasses 'breast cancer' and 'prostate cancer', it will send as articles related to 'cancer' all those that it has analyzed where appears 'cancer' but not 'breast cancer' nor 'prostate cancer'.

When the builder agent receives the information of the searcher agent, destroys it and analyzes one by one the new classes that it has received, discarding those that are equivalent to any class that it received previously, for any of the following reasons:

- Because they contain the same words but in different order.

- Because they are distinguished only by words from the same family (with the same root). This check is made with the aid of the Snowball tool.

- Because they are distinguished only by words that are synonymous, in any of its different meaning as noun or in any of its different meaning as adjective. This check is made with the WordNet tool.

After making the checks, the builder agent adds to the ontology the definitive subclasses, the relationship between each of them and their parent class, and the articles related to each one.

Then, the builder agent generates so many searcher agents as new classes have been created, and it assigns one of these classes to each one (in our example, two searcher agents are generated, one that will take charge of searching 'breast cancer' and the other will take charge of searching 'prostate cancer'). In this way, we return to the point in which each searcher agent searches its assigned class in PubMed, gets the text of the articles, analyzes it, etc.

The process finishes when we arrive at the level of depth fixed by the user or at a level in which we are not able to make a new decomposition.

D. Agent communication

The protocol used for the communication (transmission of messages) between the builder agent and the searcher agents, is the FIPA-Request protocol. This is the protocol that fits best with the needs of the process, since the builder agent has to send a message to each searcher agent with the action that it must perform and the required parameters for that, and later the builder agent has to receive the result of this action. And this protocol is used exactly for that, to request an action to an agent and get a result.

The builder agent initiates a FIPA-Request protocol with each searcher agent sending it a Request message and asking as service that it execute the action SearchWord, that is to say, that it perform the search in PubMed and the analysis of the articles.

When the searcher agent receives the Request message, it checks whether it understands the content and whether it can perform the required service (if the action that contains is SearchWord). If everything goes well, it sends an Agree message to the builder agent. Finally, when the searcher agent has made all its work, it sends an Inform message to the builder agent with the result of the action, that is to say, with the required information to build the ontology.

The builder agent also communicates with the AMS agent in order that it creates or kills one of the searcher agents. The protocol used in this case is also the FIPA-Request, since an agent asks another agent to perform certain service.

This process will be just like the previous one: the builder agent initiates the FIPA-Request protocol and sends to the AMS agent a Request message asking as service that it performs the action CreateAgent or KillAgent, depending whether it wants that it creates or kills a searcher agent.

Then, the AMS agent sends an Agree message to the builder agent whether it has understood the message, and

when it has performed the service, it sends an Inform message without additional information.

V. CONCLUSIONS

After making several tests and checking the obtained results, we can state that the objective of the partial project has been reached. It is possible to organize some reliable medical information existing in Internet with the help of the PubMed database.

PubMed, obviously, has been essential in the development of the project, since all the work is based on it.

The Entrez Programming Utilities have been essential as well, because with them it has been possible to search in PubMed and obtain the information.

HTMLParser has been very useful to extract the text of the web pages without labels to analyse it in an easier way.

The Lucene Analyzers are useful tools, specially the StopAnalyzer in this case. Before using this tool the stop words were accepted in the process and the results were not good and this filter improved them a lot.

The work of Snowball is very interesting and complicated, but in this project this tool has not been so useful than the others because there have not been many cases of words from the same family. In spite of this, we have used it to slightly improve the results.

WordNet has been used in two different tasks. It has been very useful to determine if a word is a noun or an adjective, because the fact of choosing only these kinds of words has been really important in the process and in the results obtained. On the other hand, it has not been so useful to find synonymous words, because there have not been many cases, but like in the case of Snowball we have preferred to use it to slightly improve the results.

OWL API has been essential to build the ontology in OWL language.

Finally, Protégé has been useful to check the results before the graphical user interface has been implemented.

This paper has described the design and implementation of info-collecting and ontology building multi-agent system. It will become an integral part of the ADAM architecture. We have made advantage of the essential properties of multi-agent systems, namely autonomy, reactivity, heterogeneity of individual agents, and integration of legacy systems.

There are a number of issues that require further investigation. For example, the agents should be able to adapt and learn from the social interactions they experience. Agents should learn which acquaintances give reliable results in which circumstances. Based on this knowledge they should be able to adapt their selection appropriately. In case of Internet search, for example, they can learn the user profile and try to update previously retrieved information automatically without direct intervention of the user.

ACKNOWLEDGMENT

L. Lhotská thanks L. Prieto for the implementation of the described multi-agent system and performing a set of experiments that have proved the functionality.

REFERENCES

- [1] Baujard O., Baujard V., Aurel S., Boyer C., and Appel, R.D. (1998): 'MARVIN, a multiagent softbot to retrieve multilingual medical information on the Web', *Medical Information* **23**, pp.187-191
- [2] Larsson J. E. and Hayes-Roth B. (1998): 'Guardian: An Intelligent Autonomous Agent for Medical Monitoring and Diagnosis', *IEEE Intelligent Systems and their Applications*, **13**(1), pp.58-64
- [3] Decker K. and Li J.(1998): 'Coordinated hospital patient scheduling', Proceedings of the Fourth International Conference on Multi-agent systems, ICMAS-98, Paris
- [4] Huang J., Jennings N.R. and Fox J. (1995): 'An agent-based approach to health care management', *Applied Artificial Intelligence* **9**, pp.401-420
- [5] Hayes-Roth B. (1985): 'A blackboard architecture for control', *Artificial Intelligence* **26**, pp.251-320
- [6] Knublauch H., Rose T. and Sedlmayr M.(2000): 'Towards a multi-agent system for proactive information management in anesthesia', Proc. of the Agents-2000 Workshop on autonomous agents in health care, Barcelona
- [7] Marik V., Pechoucek M. and Stepankova O. (2002): 'Organization of social knowledge in multi-agent systems', *Integrated Computer-Aided Engineering* **9**, pp.1-12
- [8] Lhotska L., and Stepankova O. (2004): 'Agent architecture for smart adaptive systems', *Trans. of the Inst.of Meas. and Control*, **26** (3), pp. 245-260
- [9] Lhotská, L.: Design of a Decision Support System in Cardiology. In *The 3rd European Medical and Biological Engineering Conference - EMBEC '05*. Prague: SBMILI ČLS JEP, 2005, vol. 11, ISSN 1727-1983.
- [10] FIPA: Agent Management. Internet site address: <http://www.fipa.org>
- [11] Bellifemine F., Poggi A., and Rimassa G.(2000): 'Developing multi-agent systems with JADE', *Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, Boston, MA
- [12] PubMed. Internet site address: <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed>
- [13] Entrez. Internet site address: http://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html
- [14] HTMLParser. Internet site address: <http://sourceforge.net/projects/htmlparser/>
- [15] Lucene. Internet site address: <http://www.apache.org/dyn/closer.cgi/lucene/java/>
- [16] Snowball. Internet site address: <http://snowball.tartarus.org/>
- [17] WordNet. Internet site address: <http://wordnet.princeton.edu/obtain>
- [18] OWL. Internet site address: <http://www.w3.org/2004/OWL/>
- [19] Protégé. Internet site address: <http://protege.stanford.edu/download/download.html>