On the Implementation of a Multi-Agent System for Computer Based Medical Education

A. G. Triantis[1]	PhD Student	triantis@math.upatras.gr
A.D. Kameas[1]	PhD Computer Engineering & Informatics	kameas@math.upatras.gr
G. Nikiforidis[2]	Professor	gnikif@med.upatras.gr

1 Educational Software Development Laboratory, Department of Mathematics, Univeristy of Patras

2 Department of Medical Physics, School of Medicine, University of Patras

Abstract

Although several agent-based systems now exist on the network, most are essentially centralized on a single agent. However, the distributed, large-scale, dynamic nature of many problem spaces such as Medical Education calls for open, flexible and scalable solutions. This work proposes and implements a multi-agent based system that teaches a subject of Orthopedics. The proposed solution supports easy upgrade of the system during runtime and easy adaptation of the current system to teach other medical subjects. This paper deals with the chosen agent organization architecture and the inter-agent communication mechanism, which are the keys for the development of a modular, open, and adaptive solution.

Keywords: Inscrutability, Organization of agent societies, multi-agent teams, multi-agent communication, CORBA.

1. INTRODUCTION

In this paper, we discuss the architecture and implementation of a Multi-Agent System that teaches a particular subject of orthopedics called "outlet" impingement, which is a cause of chronic shoulder pain. This work consists a pilot study on the introduction of agent-based tutoring systems in Computer-Based Medical Education (CBME).

In general, Medical Education attempts to transfer to the students a large volume of knowledge that is constantly growing. Moreover, contemporary Medical Education aims at guiding students into acquiring a set of critical problem-solving skills. Traditional Medical Education adopts a system-oriented approach: each system of the human body is examined from several perspectives of medical interest. Thus, the successful application of medical skills calls for a synthesis of different knowledge sources, which sometimes offer overlapping or incomplete information. On the other hand, any CBME software would deal not only with distributed, and dynamically changing knowledge, but also with the challenge of deploying educational services over the infrastructure of the organization.

We adopted a similar, knowledge synthesis approach in developing OPD-Tutor, aiming to investigate the potential of introducing multi-agent systems in CBME. We believe that multi-agent systems can be used to directly represent the knowledge synthesis and decision making procedures used by contemporary Medical Doctors. In this paper we present OPD-Tutor, a decentralized system, composed of “medical agents”, each of which implements a medical specialist system. A medical specialist system can be regarded as a representation of a medical expert: it contains the knowledge and is aware of the information resources, which are associated with a specific domain of medical science. In the case of CBME, the domain that the systems are aware of becomes a cognitive domain that can be used to describe (teach) a part of a generic medical subject. In essence, each specialist system can contribute a part of knowledge or an approach in different parts or cases of the subject. Each aspect of the medical subject is taught by a specialized medical agent, who is responsible both for acquiring and for

delivering the information pertinent to the subject, according to his specialty. In a companion paper [12] we discuss the generalization of the proposed architecture.
The main idea underlying our approach is similar to the one behind [6], [7], and [1]. Although these systems deal with health-care management and patient treatment, they also use a multi-agent architecture to deal with the distributed, multi-disciplinary and uncertain nature of medical knowledge. Our design focuses mainly on issues concerning the implementation and deployment of the multi-agent system, as well as issues regarding the retrieval and synthesis of medical knowledge with respect to concrete instructional plans. In this respect, it is complementary to Adele [10] a pedagogical agent, which comprise mechanisms presentation, student monitoring and multi-modal interaction with a student through feedback, probing questions, hints, and explanations. This agent paradigm has been applied on teaching time-critical trauma exercise.
The tutoring architecture of OPD-Tutor is described in section 2. In the next section, we describe inter-agent communication, without getting into technical details. In section 4, we present implementation issues that we faced during development; they include the message structure, the agent internal architecture and selected technology for the implementation of the communication mechanism. Finally, the conclusions drawn from the development of OPD-Tutor, along with our next steps are discussed in section 5

2. OPD - Tutor Tutoring Architecture

We approached the CBME problem in a manner that is common among physicians: each specialist in a medical subject contributes his knowledge on the case at hand, in order to solve as a team a problem that is beyond the individual capabilities and knowledge of each member of the team. As a consequence, OPD-Tutor consists of a society of autonomous agents, each of which is a medical expert.
The case at hand (teaching the subject of outlet impingement) requires the contribution of the following medical experts: Orthopedist, Anatomist, Kinesipathist, Radiologist, Pathoanatomist, Pathophysiologist, Laboratorian and Physiatrist, each of which is a medical expert in the Orthopeadics, Anatomy, Kinesipathy, Actinology, Pathoanatomy, Pathophysiology, Laboratory, and Physiatry, respectively. In the multi-agent organization, each medical expert contributes its specific knowledge, capabilities, responsibilities and behavior; each expert's role inside the organization is determined by these characteristics.
In general, an organization populated by agents provides a framework for agent interactions through the definition of roles, behavior expectation, and authority relations. In [11], Sycara points out four kinds of organizations: the *Hierarchy*, where the decision control belongs to a single agent and interaction takes place through vertical communication; the *Community of Experts*, where a group of specialists interact through rules of order and behavior; the *Market*, where a group of agents interact through bidding and contractual mechanisms; and the *Scientific community* where solutions to problems are published for testing and refinement.
In designing OPD-Tutor, we adopt a hybrid organization (Figure1) that combines features from the first two approaches. In order to collect and synthesize medical knowledge, a group of medical specialists agents is formed according to the *Community of Experts* organization. However, even though we permit vertical *and* horizontal communication we do not adopt the decision control to be based on bidding and contractual mechanisms. On the contrary, decision control belongs to a single agent, as in the *Hierarchy* organization.

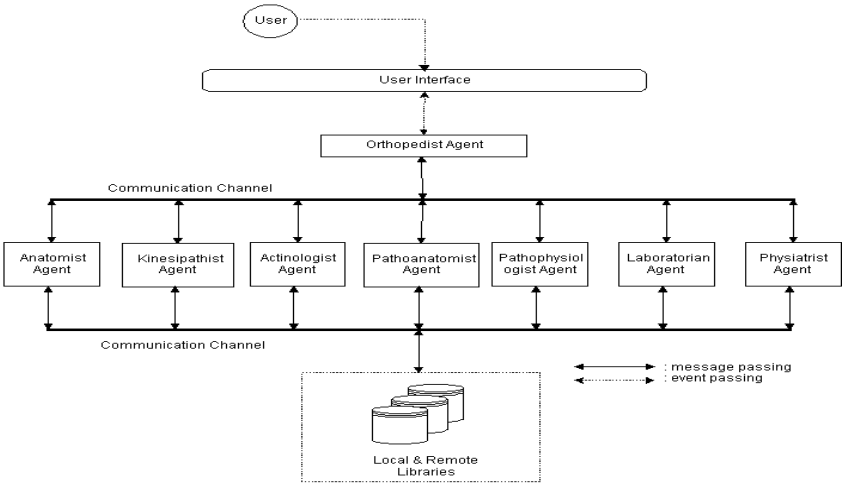

Figure 1: OPD - Tutor Tutoring Architecture

Bearing in mind that this architecture is used for tutoring purposes, only the *Orthopaedist Agent* can act as an instructor; all the instructional strategies necessary for the tutoring process of "outlet impingement" are contained in this agent's plans. We approached the CBME problem using the previous architecture for two reasons:

1. Generally, the teaching process of a subject is based on a specific teaching strategy, which is known to one agent (instructor). The teaching strategy may of course consist of different teaching plans
2. Since no single agent can possess all medical knowledge, it is distributed across many medical agents, who have to cooperate under the guidance of the instructor, in order to retrieve and present those pieces of knowledge that match the instructor's plan

3. OPD – Tutor Low Level Architecture

In this section we describe the low-level architecture of OPD-Tutor (Figure 2) and particular the inter-agent communication structure. In order to achieve modular, open and dynamically changing behavior of the system, the proposed architecture uses an agent called *Facilitator* [5]. Facilitator is not involved in the teaching procedure. This is the reason why Facilitator is not shown in the high level architecture of the educational system. The role of Facilitator is to provide "yellow pages" services to other agents. In other words, Facilitator is responsible for maintaining an accurate, complete and timely list of the services that a registered agent can provide. When each agent is started, his first responsibility is to register himself and his services to Facilitator.

This is a key action supporting the openness and the distributed behavior of the system, because its configuration does not depend on any particular agent and it may change depending on the instantiated agents. Each agent that participates in the organization is implemented as an individual, autonomous program without either knowing the capabilities or the existence of the other agents, except Facilitator.

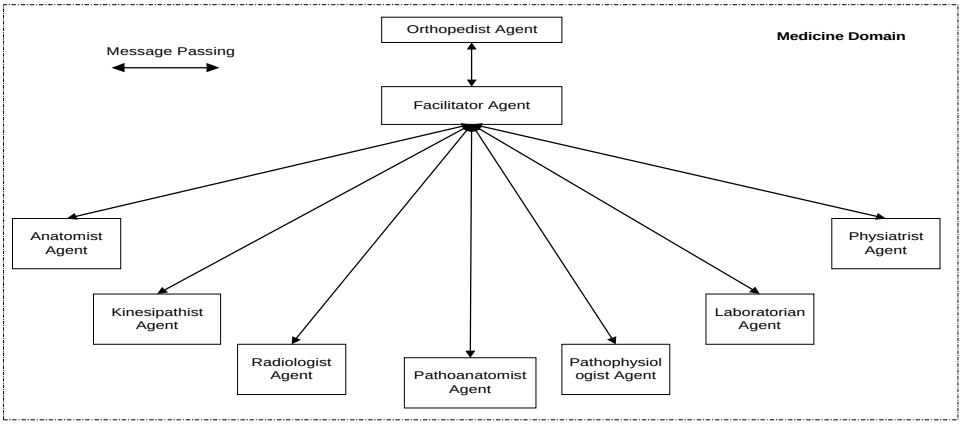

Figure 2: OPD – Low Level Architecture

All the communication that takes place among the agents passes through the Facilitator. For example, if the Orthopedist Agent wishes to send a message *a* to the Radiologist Agent, he will first send a message *b* to the Facilitator asking for him to forward the message *a* to the Radiologist Agent. The answer of the Radiologist Agent will pass through the Facilitator, too.
Allowing point-to-point communication only between an agent and the Facilitator, provides OPD-Tutor with desirable properties such as:

1. Modularity. Each agent can be implemented by different people and different programming tools with the restriction of knowing the communication means (explained later) between him and the Facilitator of the OPD-Tutor.
2. Reusability: It permits an agent to be member of more than one CBME systems, each of which may deal with different Medical Education domains.
3. Adaptability: It is easy to change the instructional strategy and the education methodology of OPD-Tutor simply by substituting one instructional agent with another, without having to re-program and recompile the whole system. However, the current architecture does not provide for more than one instructional agents. Such an extension would require negotiation among different instructor agents, probably, on the basis of which instructional strategy is more suitable for the current user.
4. Openness: It is possible to replace any medical agent "on –the-fly" (dynamically and transparently), while the system is in use. For example, the Anatomist agent can be replaced by another Anatomist agent, which is implemented probably by different programmer(s), and includes different plans[1].
5. Plug 'n' play. By registering his capabilities to Facilitator, when each agent is started, he enters the tutoring processing upon instantiation.
6. Scalability. Considering that a Facilitator controls one educational domain (Medicine in our case), the system can easily be scaled up to support multi-domain education by adding one or more multi-agent systems, each with its own Facilitator which interact through a "common" Facilitator of a "higher" level.

The main disadvantage of point-to-point communication only between an agent and the Facilitator is the unavoidable delay of message transportation from one agent to another. The delay increases if more than one domain participate in the tutoring architecture, since communication may pass through the Facilitator of two domains. Nevertheless, the above-mentioned advantages and the speed of contemporary computers make up for these delays. The above advantages of the selected low-level architecture fulfilled the goals of this framework.

[1] Plans describe the possible ways that an agent can bring about a goal

4. Implementation Issues

In our work we consider an agent to be a modular and autonomous executable program. His implementation comprises the development of three separate parts: the Message Structure, the Agent Core and the Communication Module (Figure 3).

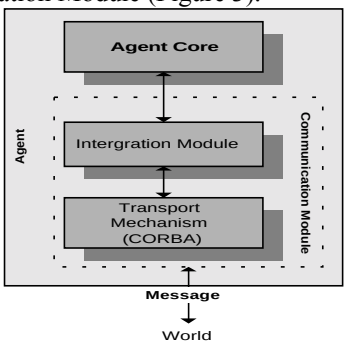

Figure 3: The agent structure

4.1 The Message Structure

A message is an individual unit (building block) of inter-agent communication that is based on the speech act theory [9], where every intended communication action from an agent changes the world in an analogous way a physical action does. A message as defined by FIPA in [4] and implemented in OPD-Tutor system, has the following structure:

```
Communicative act (
                    Sender
                    Receiver
                    Content ()
                    in-reply-to
                    reply-with
                    Language
                    Ontology
                )
```

Figure 4: Message Structure

The first word of the message structure represents the communicative act (CA) of the message (corresponding to a performative of KQML). An agent performs an action, which is implied by the CA having as parameters the elements of the message structure as shown in Figure In our framework, we make use of a subset of FIPA's CAs, which are common for all the agents that populate the society of the medical educational system. Each CA implies that the content of the message contains a specific set of actions. In this case,

- The "cancel", "request" and "refuse" CA implies that the receiver will perform a specific action.
- The "inform" CA implies that the receiver will update his knowledge base and especially his believes about specific things of the environment.
- The "failure" and "not-understood" CA implies that the receiver will update his knowledge base and will act in order to handle the error handling.

The comprehension and the execution of the action depends on what actions the recipient can perform. These actions are defined by the *Content* element, which has precisely defined syntax, and semantics [4]. In our framework, we make use of the PROLOG language structure and syntax, in order to define the *Content* element. As a consequence, each action is a PROLOG data structure.

The *Content* is not the same for all the agents, but depends on the actions that each agent performs. In this case, the actions are divided into those supported by Facilitator and those supported by either of the medical agents or the Orthopedist agent.

4.1.1 Actions supported by Facilitator

Register: An agent registers its services to the Facilitator in order to publicize some or all of them to other agents. There is *no* intended future commitment or obligation, on the part of the registering agent implied in the act of registering. For example, an agent can refuse a request for a service, which is advertised through the Facilitator. There is a commitment on behalf of the Facilitator to honestly broker the information it holds.
Search: A search action implements a request from an agent, who expects to get a specific list of agent names that involve specific services. The Facilitator is responsible for searching his database of the registered and active agents and to provide the list of the agent names. The answer to a *search* request is a *result* information.
Modify: This action involves changing an agent's registration fields. The content of a modify message will replace that information which is currently registered for that agent.
*Deregister***:** An agent de-registers in order to remove any record of his services from a domain. The de-register action has the consequence that there is no longer a commitment on behalf of the Facilitator to broker information relating to that agent.
Forward: An agent can ask the Facilitator to forward a message to a destination agent. In the case that the sender does not know the name of the receiver then Facilitator has to search among the registered agents with the service as a keyword and forward the message to one of the agents (randomly) that provide the same service. For example, when the Orthopaedist Agent needs someone with knowledge of Anatomy, he requests from the Facilitator to forward the message to a registered agent who provides Anatomy services.

4.1.2 Actions supported by the rest of agents

*Teach***:** The teach action implies that the recipient will undertake the responsibility of finding the appropriate Learning Unit (LU)[1] for the specific plan name and return the LU's location to the sender.
*Location***:** The location action returns to the sender the appropriate LU's location according to the requested plan-name The location action is an answer to the teach one. The location action updates the sender's knowledge base.
*Result***:** The result action is an answer to the requested action search. The Facilitator returns to the sender a list of agents who can provide the requested service. The result action updates the sender's knowledge base.

4.2 The Agent Core

The Agent Core is the essential part of each agent. It contains the knowledge base and the reason-about mechanisms of each agent. Its structure is based on the well-known BDI logical framework [3] where each agent is viewed as having the three mental attitudes of belief, desire and intention (BDI). In addition to three modules that represent the three mental attitudes (Figure 5) every agent contains a plan library, an inference engine and a communication module. Plans describe the possible ways that an agent can bring about an intention. In general, a plan is a partial commitment on how to achieve a desire. The Inference engine is the mechanism that handles and updates the agent's mental state (beliefs, desires and intentions), selects, executes and rejects plans according to the mental state and processes the incoming message that transfers the observations of the agent's environment.
In our framework, we implement the Agent Core using the PROLOG programming language because we believe that the built in backtracking mechanism as well as its data structures that it supports, makes it suitable programming language for non-deterministic management of knowledge and information.

[1] A Learning Unit (LU) is a piece of Knowledge represented in multimedia format.

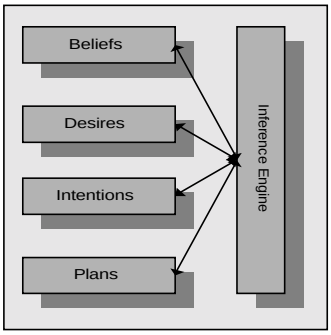

Figure 5: Internal agent architecture

4.3 Communication Module

The communication module is responsible for message transportation. It consists of two separate parts (Figure 3): the Integration Module and the Transport Mechanism.

4.3.1 The Integration Module

The integration module is responsible for the integration between the Agent Core and the Transport Mechanism. In this way, we support the ability to choose the appropriate transport mechanism according to the current implementation circumstances without having to entirely redesign the agent's implementation.

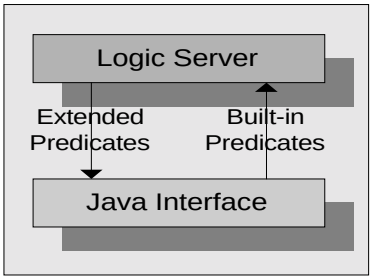

Figure 6: The Integration Module

The integration module consists of two parts (Figure 6): The Logic Server and the Java Interface. The Logic Server is the PROLOG runtime engine, which exports a collection of functions that provide access to the built-in Prolog predicates, including the ability of backtracking as well as the implementation of custom built-in predicates, called extended predicates. In our work, we make use of the extended predicate *Send* (Figure 7) in order to send a message via the transport mechanism directly from PROLOG.

```
Send (
        CA,
        Sender,
        Receiver,
        Content,
        Reply_to,
        Reply_with,
        Language,
        Ontology)
```

Figure 7: The Send Extended Predicate

The elements of the *Send* correspond to the message structure. The *Content* element can contain a whole message in the case, where an agent requests from Facilitator to forward a specific message to another agent. The Java Interface contains the implementation of the *Send* extended predicate and the transport mechanism. In addition, it is responsible for the integration of the two previous separate implementations.

4.3.2 The Transport Mechanism – CORBA

The transport mechanism is the physical means for message exchange. Several transport mechanisms have been proposed in agent literature such as TCP/IP sockets, Java RMI, HTTP, Unix RPC, Email and CORBA. In our framework, considering each agent as a Client/Server entity, the transport mechanism has been implemented with CORBA for two reasons:

1. FIPA addresses a minimal set of requirements on the communication protocol for building open, distributed systems [4], which are fulfilled by CORBA.
2. CORBA offers services, such as naming services, trader services, access control services, event services, which make it suitable to easily implement inter-agent communication mechanisms.

The CORBA implementation is described by an Interface Definition Language (IDL) file, which describes the services that a CORBA object can provide. In the context of multi-agent systems we used the IDL file (Figure 8) to describe (in the form of an interface) a procedure that takes as parameters those elements of the message structure which take place in inter-agent communication. The implementation of this interface is taking place in the Java interface (Figure 6).

```
module Agents {
interface SendManager {
void Send message( in string communicative_act,
                   in string sender,
                   in string receiver,
                   in string content,
                   in string in_reply_with,
                   in string in_reply_to,
                   in string language
                   in string ontology );
            );
        );
```

Figure 8: The IDL file

A similar work to ours (KQML-CORBA based multi-agent toolkit for management) is described in [2], where the authors propose a different CORBA IDL file. The main idea is that each of the values of the performative element is described in the IDL file as an interface. Consequently, the processing of the messages takes place in the implementation part of these interfaces. In contrast to this approach, we propose the IDL file of Figure 8 which describes only the message structure and not the values that describe the message. With this approach we manage to delegate the processing of the message to the inference engine of agent core module.

5. Discussion

During the last month of the previous academic year, the system was put to a trial operation by granting a group of medical students with access to it. In total 30 students used the software in 3 sessions of 10. Each student was using a PC (Pentium 133, Windows NT) connected to the local network and the Internet. The prototype of the application contained 9 agents (as shown in Figure 2), each of which was running on a different machine of the network. The learning units were stored in a database and managed by a database server program. Each agent was instantiating separate database clients in order to access the database. During each session, each student used the system six times, performing different tasks.

After each system run, we asked the students simple questions about the features they liked or disliked, the capabilities they would like to see and the technical problems they encountered. Although this does not constitute a complete summative evaluation (it was rather a system testing at the environment where it will be used), the initial processing of the results showed that:

Students were attracted by the following system features:

1. robustness: the system was still operating (albeit providing a limited functionality) even when a part of the network was shut down (provided that the facilitator agent and the instructional agent were operational).
2. presentation: the system presents the next tutoring action as a set of hypermedia links to be visited by the student. This allows the student to learn on his own pace.
scalability: when a new medical agent was added the students could immediately see the results of its services, as more links to learning units were available for the same tutoring steps
A set of problems has to resolved:
1. user interface must become adaptive to each student's needs. In order to achieve this, a student model must be designed. We plan to implement a user-interface agent analogous to Adele [1].
2. the response time has to be improved. Delays in presenting the next tutoring step are due to the message exchange between each agent and the Facilitator agent. We expect to reach a trade-off between complete modularity / adaptability (which is supported by including all structural knowledge in the facilitator only) and speed (by storing a part of structural knowledge with each medical agent - i.e. a set of most recent agents it contacted). Although this will allow us to implement point-to-point communication between medical agents, a degree of scalability will be sacrificed

6. Acknowledgments
Our thanks to Dr. M. G. Salmas, lecturer with the Department of Anatomy, University of Athens for the consultation he provided on the teaching of “outlet” impingement. The described work is part of the project *X-Genitor* (PENED 99ED68) founded by the General Secretariat for Research and Technology of the Ministry of Development and European Social Fund.

7. References

1. Aknine, S., and Aknine, H. (1999). Contribution of a Multi-agent Cooperation Model in a Hospital Environment. In Proceedings of the Second International Conference on Autonomous Agents ACM Press.
2. Benech, D., and Desprats, T. (1997). A KQML-CORBA based Architecture for Intelligent Agents Communication in Cooperative Service and Network Management. In IFIP/IEEE International Conference on Management of Multimedia Networks and Services '97. Montreal. Canada.
3. Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. Artificial Intelligence, 42:213-261.
4. Foundation for Intelligent Physical Agents. FIPA 97. (1997). Specification. Part 2. Agent Communication Language. Geneva, Switzerland.
5. Genesereth M. R. and Ketcpel S. P. (1994). Software Agents. Communications of ACM, 37(7): 48-53.
6. Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A. Vina, A., and Seiver. A. (1992). Guardian: A Prototype Intelligent Agent for Intensive-Care Monitoring. Journal of AI in Medicine: Special issue on Expert Systems for the Intensive Care Setting.
7. Huang, J., Jennings, R., and Fox, J.: An Agent Architecture for Distributed Medical Care, Intelligent Agents (Eds. M. J. Wooldridge and N.R. Jennings), Lecture Notes in Artificial Intelligence, Springer Verlag, 1995, 219-232.
8. N. R. Jennings and M. Wooldridge.(1998) Applications of Agent Technology In N. R. Jennings and M. Wooldridge, editors, Agent Technology: Foundations, Applications, and Markets. Springer-Verlag, March 1998.
9. Searle, J. R. Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press: Cambridge, England (1969).
10. Shaw, E., Johnson W., L., and Ganeshan, R. (1999). Pedagogical Agents on the Web. In Proceedings of the Second International Conference on Autonomous Agents ACM Press.
11. Sycara K. (1998). Multiagent Systems. AI Magazine vol 19, No 2, 78-92
12. Triantis A, Kameas A, Zaharakis I et. Al (2000) Towards a generic Multi-Agent Architecture of Computer-Based Medical Education Applications. Πρακτικά του 2ου Πανελλήνιου Συνέδριου "Οι Τεχνολογίες της Πληροφορίας και της Επικοινωνίας στην Εκπαίδευση".