Association for Information Systems
AIS Electronic Library (AISeL)

MCIS 2011 Proceedings

Mediterranean Conference on Information Systems (MCIS)

2011

DY.M.A.CO.S. – DYNAMIC MOBILE AGENTS FOR COLLABORATION SYSTEMS

Dimosthenis T. Georgiadis
University of Nicosia, georgiadis.d@unic.ac.cy

DY.M.A.CO.S. – DYNAMIC MOBILE AGENTS FOR COLLABORATION SYSTEMS

Georgiadis, T. Dimosthenis, University of Nicosia, Department of Management & MIS, 46 Makedonitissas Ave., 1700 Nicosia, Cyprus, georgiadis.d@unic.ac.cy

Abstract

For many years collaboration between colleagues was been achieved with the conventional telephone, fax and regular mail. Over the last 10 years the way that colleagues communicate with each other changed dramatically. The huge internet spawning and multi-channel evolvement contribute to this transition providing new ways of communication like video conferencing systems, e-mails, forums and other collaborating systems. In addition, mobile wireless computing brings about a new paradigm of distributed computing in which communications may be achieved through wireless networks and users can continue computing even as they relocate from one support environment to another. Equally important, however, is the design and implementation of data management applications for these systems (e.g., wireless virtual teams) a task directly affected by the characteristics of the wireless medium, the limitations of mobile computing devices, and the resulting mobility of resources and computation. eHealth is a discipline that took over in the last years, establishing the need for providing dynamic working environments of different actors (medical professionals) promoting effective collaboration among the actors, including the patient, at any time, and any context (place). In this paper we will present a toolkit that creates virtual teams for collaboration systems with the use of mobile agents in eHealth.

Keywords: Mobile Agents, Collaboration, Dynamic Virtual Teams.

1 INTRODUCTION

Over the last years, companies tend to expand over boarders making the need for tele-collaboration imperative. The conventional methods like mail, fax and telephones proved not to be enough to support these needs. Now, the huge technological leaps supports other methods for collaboration more efficiently, been productive and cost effective. Many collaboration systems spawned just to cover this need using these newly available technologies.

An extensive study on collaboration systems and schemas showed us that all these systems are using many common features such as messaging, groups and virtual teams (Pitsillides A. et al., 2005). It would be useful if there was a system that it could provide collaboration features easily, fast and with practically no expense giving the ability to be adopted to any application that we want.

Besides the functional components of a mobile application, the organization of data is also important. Data may be organized as collections of objects. In this case, objects become the unit of information exchange between mobile and static hosts. Objects encapsulate not only pure data but also information regarding their manipulation, such as operations for accessing them. Incorporating active computations with objects and making them mobile leads to mobile agents.

Mobile agents are processes dispatched from a source computer to accomplish a specified task (Chess et al., 1995; J.E.White, 1996). Each mobile agent is a computation along with its own data and execution state. In this sense, the mobile agent paradigm extends the RPC (Remote Procedure Call) communication mechanism according to which a message sent by a client is just a procedure call. After its submission, the mobile agent proceeds autonomously and independently. When the agent reaches a server, it is delivered to an agent execution environment. Then, if the agent possesses necessary authentication credentials, its executable parts are started. To accomplish its task, the mobile

agent can transport itself to another server, spawn new agents, or interact with other agents. Upon completion, the mobile agent delivers the results to the sending client or to another server.

A general mobile agent model usually supports disconnected operation. During a brief connection service, a mobile client submits an agent to the fixed network and then disconnects. The agent proceeds independently to accomplish the delegated task. When the task is completed, the agent waits till reconnection to submit the result to the mobile client. Conversely, a mobile agent may be loaded from the fixed network onto a laptop before disconnection. The agent acts as a surrogate for the application allowing interaction with the user even during disconnections. Weak connectivity is also supported by the model since the overall communication traffic through the wireless link is reduced from a possibly large number of messages to the submission of a single agent and then of its result. In addition, by letting mobile hosts submit agents, the burden of computation is shifted from the resource-poor mobile hosts to the fixed network. Mobility is inherent in the model. Mobile agents migrate not only to find the required resources but also to follow mobile clients. Finally, mobile agents provide the flexibility to load functionality to and from a mobile host depending on bandwidth and other available resources. And thus, materializing high degree of adaptivity. (Samaras G. et al. 2005)

In this paper we propose DY.M.A.CO.S. (Dynamic Mobile Agents for Collaboration Systems), a system that creates dynamically collaboration systems with the use of mobile agents. All the capabilities that mobile agents have are inherited to our collaboration system, such as autonomy, persistency and ubiquity. Our system can be used without any software installation, using only a simple html browser. In section 2 we will describe in short the background theory (section 3), following the description of the system (section 4) and finally a case study with the conclusions on this work (section 5).

2 BACKGROUND THEORY

2.1 Mobile Agents

Mobile Agents are programs or more correctly Objects (usually in JAVA), which are “running” on a computer having the ability to move to another computer, continuing their execution. Usually, Mobile Agents have the following features.

- **Mobility:** agents can move from one computer to another keeping their data and state during movement.
- **Autonomy:** after their creation, agents can execute their tasks without their creator’s supervision.
- **Communication:** agents can communicate with their owner, other agents and even with their execution environment anytime.
- **Fault Tolerance:** their functionality does not get affected by any network failure.

Mobile agents are small in size and they cannot be an autonomous application. They can collaborate though with other mobile agents and then they formulate an autonomous application.

They can be used in many applications like:

- Distributed Databases
- Artificial Inteligence
- Distributed and Parallel Computation
- Distributed Algorithms
- Teleworking
- Virtual Enterprise

There are many technologies for deploying mobile agent systems. The most common technologies are the Concordia, the Grasshopper, the Aglets and the Voyager (Harrison G. et al., 1995; Mitsubishi Electric ITA, 1998; Lange D. et al., 1996; Lange D. et al., 1998; Dikaiakos M. et al., 2001; Samaras G. et al., 1997).

Concordia is a mobile agent development environment from Mitsubishi Electrics, based on Java. It uses the services of many manager objects to provide a complete framework for development and management of the mobile agent applications. Concordia supports service naming for applications and agents. The most important minus of the Concordia technology is that these agents cannot get through firewalls when the travel through PPP connections (Concordia by Mitsubishi).

The Grasshopper platform is a mobile agent development environment of GMD FOKUS and IKV++, which provides dynamic CORBA support. These agents have the ability to move autonomously between servers. This movement is succeeded with the pre-installation of the Distributed Agent Environment (DAE). The DAE is an environment in which the agents are executed. The Grasshopper technology does not provide any Manager, so it is on developer's job to create a Manager object (Grasshopper by GMD FOCUS).

The Aglets Workbench is a mobile agent Java environment from IBM Japan research group with the name "Aglets" (agile applets). An aglet is a Java object, which can be sent to another computer and continue his execution on the new server. Aglets, during their movement, carry their code, data and state with them continuing with their execution uninterruptedly. Aglets are autonomous and their strong point is that they are able to carry with them their itinerary as a java object. During their travel, they can change dynamically this itinerary. Aglets can communicate with each other with massages that are been exchanged as a java objects (Aglets by IBM).

Voyager is a mobile agent development environment from ObjectSpace, which supports CORBA. Developers can use Java classes as server objects without changing the source code. The Voyager's architecture uses internally the Java Beans Event model for notify an application on the system and object layer. The object events are associated with specific objects of the virtual machine. The developers can build listeners for tracking the virtual machine's traffic and act in proportion according the events that are occurred. A special category of listeners are the Assistants who are strictly attached with an object. With the term strictly we mean that if an object moves to another computer, its assistant listener moves along. Finally, voyager has an option for a security manager to monitor the agents that are accessing the host. It also reduces the security constraints for applets and allows objects which are on the browser's side server to communicate with objects that are not on this server (Voyager by Object Space). In our system, the voyager platform was selected in order to utilize the mobile agents' notion.

2.2 Virtual Teams

The idea of the virtual team is not yet clearly defined and many times its definition is confused with the one of virtual workspace or the one of virtual communities and teleworking.

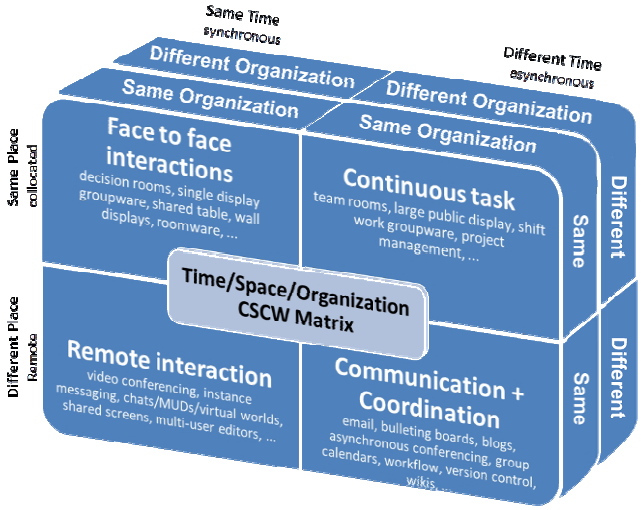

Figure 1. All possible types for virtual teams.

In general, virtual teams are "*Geographically dispersed and organizationally interdependent entities whose members collaborate through enabling technologies*". (Pitsillides A. et al. 2005) A further refinement of this characterization is by imposing five additional qualifying criteria: (1) members work apart more than they do at the same location, (2) they rely on communication technologies such as computer networks, email, groupware, and phones, (3) members solve problems and make decisions jointly, (4) members are mutually accountable for results and (5) teams members in such organizations can be brought together for a finite time to tackle specific tasks or problems.

The team members can work for the same or different organization, can be or not in the same place and co-exist at the same or different time (Figure 1). These eight cases are all the possible combinations that can be occurred in a virtual team.

In the creation of a virtual team, some important issues of concern are the ones of trust and identity. It is natural to have persons in a virtual team that they are separated, thus there is a problem of certify that the other member is indeed the one that is supposed to be. One simple way to authenticate a user is the use of username and password.

2.3 Collaboration Systems in eHealth

Hospital based treatment for chronic patients is limited, often demanding-based for short periods of time, used mainly for acute incidents. As it is not possible for the health care team to be physically present by the patient at all times, or at any time physically together, whilst the patient is undergoing treatment at home (or work), a principal aim is to overcome the difficulty of coordination and communication. Various eHealth systems, such as DITIS (ΔΙΤΗΣ, in Greek, stands for: Network for Medical Collaboration) provide effective and efficient collaboration among health specialists. In addition, DITIS support dynamic Virtual Collaborative HealthCare Teams dealing with the home-healthcare. It supports the dynamic creation, management and co-ordination of virtual medical teams, for the continuous treatment of the patient at home, and if needed for periodic visits to places of specialised treatment and back home. DITIS system integrates a number of state of the art technologies to provide a sophisticated tele-application in the medical domain. To substantiate the importance of such system one must differentiate it from other related attempts. Indeed a number of related projects and efforts exist; however, no one, as far as we can ascertain, offers the features and innovation of DITIS within a single integrated application. Indicatively, these systems can be classified based on their focus activities. (Pitsillides A. et al. 2005)

There are various projects covering accurate diagnosis and treatment until a patient gets to the hospital or in a home care center. Among them are: Ambulances Health Telematics HC 1001, Emergency-112 Health telematics HC 4027, Vital-Home ISIS 98 502085) that have already developed such telemedicine services over GSM. However these have not addressed the notion of virtual collaborative medical teams for the home care sector. The projects that deal with home care can be divided in two types: The first type is closer to research projects concerned with the delivery of home care developing new generation of bio-sensors and transmitters (e.g. CHS, CHRONIC, EPIDEMICS, M2DM, TELEMEDICARE, E-REMEDY). The second type improves collaboration between health professionals for better home care delivery (e.g. D-LAB, PHARMA, MTM, MOEBIUS). None of these projects consider mobile technologies to support health-care team collaboration in home care applications. (Pitsillides A. et al. 2005)

Similarly a number of European projects are actively focusing on the use of handheld devices for health care provision. WARD IN HAND (IST-1999-10479) allows the management of key clinical information while providing decision support to mobile medical staff of a hospital ward, MOBIDEV (IST-2000-26402) promises to provide mobile users with secure access to the Hospital Information System in and outside the hospital, using web interfaces based on Bluetooth technology and GPRS/UMTS networks and also improve user friendliness via voice commands, DOCMEM (IST-2000-25318) and MOMEDA (HC 4015) aim to offer web access to electronic patients records (EMR) via multimedia terminal and possibilities of remote consultation, SMARTIE (IST-2000-25429) aims to develop web tools for multi-platform EMR access and support for medical error prevention, MTM (IST-1999-11100) provides via a local wireless network multimedia medical support to the mobile

hospital personnel. These projects are based on a variety of technologies (e.g., GSM, GPRS or local wireless networks) and face the mobile health care problem from a variety of angles. To our knowledge, none of the above considers collaboration and virtual healthcare teams using commonly available GSM/GPRS based communication channels and handheld devices (e.g. Smart Phones) utilizing mobile agents technology. (Pitsillides A. et al. 2005)

3 SYSTEM ARCHITECTURE

DY.M.A.CO.S. system can divided into 2 phases. The first phase is the phase of the collaboration system creation and the second one is the phase that the users collaborate with each other using the result of the first one. In general both phases (creation and execution) have almost the same architecture (Figure 1, Figure 2). Through a single Servlet (Communication Servlet – C.S.), user accesses the proper agent to complete his task.

The components that are used in the DY.M.A.CO.S. system are:

- **Manager Agent (M.AG.):** Creates the new collaboration systems and the agents.
- **Communication Servlet (C.S.):** Establishes communication between users and agents.
- **Custom Java Libraries (C.J.L.):** Libraries that are used by M.AG. for creating new mobile agents and collaboration schemas.
- **System Database (S.DB.):** Repository for the collaboration schema data.

During the first phase (creation) there is only one agent, the Manager Agent (*M.AG.*). After the user inserts the data of the collaboration schema that he desires to create (through *M.AG.*), the first phase is completed, resulting to a system that will be used on the second phase (execution).

In more detail, during the first phase, the user forwards his requests to the *C.S.* through a simple HTML browser. The servlet forwards these requests to the *M.AG.* and this agent after reading the Custom Java Libraries (C.J.L.) creates the new collaboration system and updates the System Database (S.DB.) (Figure 2).

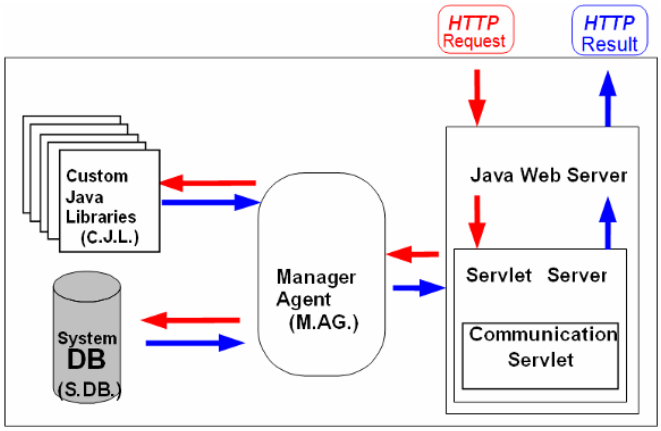

Figure 2. First Phase Architecture (Creation) [1]

After the creation of the new collaboration system, users access through *C.S.* their agents in the Mobile Agents Collaboration Space (M.A.CO.S.). The agents collaborate with each other or access the

[1] ***M.AG.:*** *Manager Agent for the creation of the final system (result)*

S.DB.: *System Database for storing the collaboration systems*

C.J.L.: *Custom Java Libraries for creating the final system's mobile agents*

C.S.: *Communication Servlet connects users with proper agents*

S.BD. or an Application Specific Database (A.S.DB.) in order to produce some results. These results are returned to the end-user through the *C.S.* (Figure 3).

A better understanding of the DY.M.A.CO.S. system we present the following scenario. A user connects with the system to create a collaboration system. The data flow goes as follows: (i) user through a browser communicates with the *C.S.*. (ii) The data are formed and forwarded to the *M.AG.*. (iii) *M.AG.* reads the *C.J.L.*, updates the *S.DB.* with users and all the necessary data for the collaboration needs and then (iv) creates the mobile agents that compose the new collaboration system (Figure 2). (v) The new collaboration system is consists of the new mobile agents (M.A.CO.S.), the *S.DB.*, the *C.S.* and the *A.S.DB.* (Figure 3). The *A.S.DB.* is different according to the users' needs and sometimes it may not even exist in the result of the new system. In the following section we present all these technologies in more detail.

3.1 Communication Servlet (C.S.)

The basic idea is that users can connect through a simple browser with the system and communicate with other users and transfer data according to their application though HTML forms. The main problem was to locate and manage the proper agent. There is no way that a browser can communicate with an agent platform without any extra installation like an applet.

We overcome this problem with the use of the Communication Servlet (C.S.). The *C.S.* is a simple servlet, named "MainServlet.class", that does not do any computation on user's data. The tasks that it handles are to collect the information from these HTML forms, format them according to the messaging protocol (see 3.4) and forward them to the proper agent.

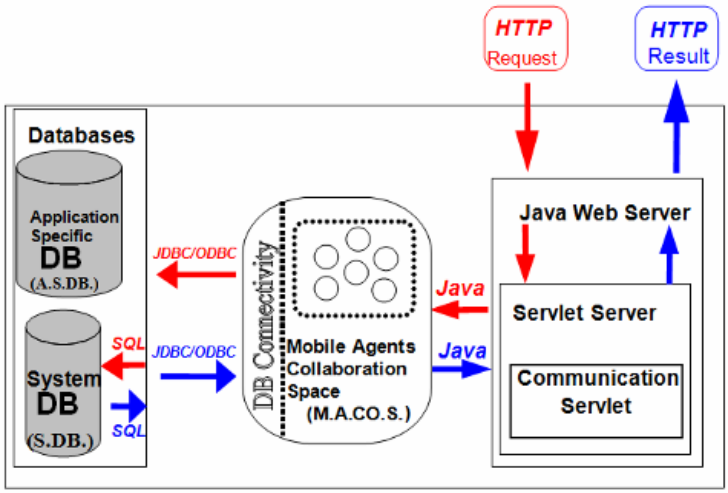

Figure 3. Second Phase Architecture (Execution)[2]

3.2 Manager Agent (M.AG.)

When a new collaboration system is built, users have no personal agents to communicate; therefore we have a general purpose agent. This agent is the Manager Agent (M.AG.). The *M.AG.* is the first and the only agent that exist in the initial DY.M.A.CO.S. system. All other agents are created or invoked by M.AG. In general, the tasks that this agent handles are:

- **Creation of new Agent Types:** Manager Agent creates the new agent type (Figure 4) after receiving many questions for the new collaboration system and after accessing the Common Java Libraries (see 3.3). These new agent types along with the communication part, they will perform the tasks needed by the application context.

[2] ***M.A.CO.S.:*** *Mobile Agents Collaboration Space*

A.S.DB.: *Application Specific Database*

- **Creation of New Users:** When a new user creates a new agent (instance, not type) the system asks him to insert username, password and the agent type along with some other information.
- **Simultaneously Creation of New Agent Type and User:** The previous two functions can be done simultaneously if while the use declares the new agent type, inserts also some user values such as username and password.
- **Existing Agents Invocation:** For each user that have an agent in the system, the *S.DB.* stores some useful information such as username, password, location and agent type. If for any reason the system stops, we do not want this information to be lost. Another *M.AG.*'s task is to invoke all the existing agents from the *S.DB.* (Figure 5).

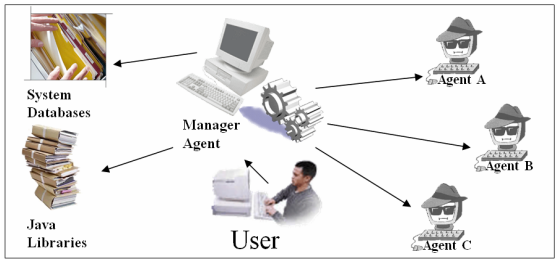

Figure 4. Creation of New Agent Types

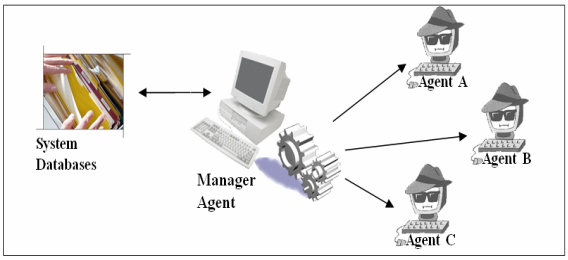

Figure 5. Existing Agents Invocation

3.3 Custom Java Libraries (C.J.L.)

After analysing the needs of a complete collaborating system we noted some widely used functions that employed for collaboration. These functions were developed in Java and enriched our Custom Java Libraries. In other words, the *C.J.L.*'s contain Java code that can be used from agents that want to have specific collaboration functionality.

A typical example of a Java program follows this structure:

```
1.  Agent.java
2.  import java.lang.*;
3.  …….
4.  /* Users Code 1 Here*/
5.  public class Agent implements Interface, Serializable
6.  {public String name = null;
7.  ……
8.  /* Users Code 2 Here */
9.  public Agent()
10. {super();}
11. …..
12. /* Users Code 3 Here */}
```

We can see that a developer can add any code in point 4 (Code 1), parameter in point 8 (Code 2) and more functions in point 12 (Code 3) allowing the development of agents with any possible desirable functionality.

3.4 Messaging Protocol

In DY.M.A.CO.S. users communicate with the system through HTML forms. The *C.S.* does not know the exact number and names of the HTML form fields. If it asks for a wrong field name, the servlet will crash terminating the system's operation.

The only solution to this problem was to set a messaging protocol. The protocol that it was adopted dectates that all the form fields must have names like req0, req1, req2,…. , reqN. The field req0 determine the number of the fields that the servlet must read. The field req1 determine the name of the agent that this information must be forwarded. So, *C.S.* only reads the fields and creates a table named request[req0,req1,req2,req3,…,reqN], where req0=N-2 and forwards it to the agent req1. Agent req1

reads the req2 and it knows what to do. The next page is created from the agent following this messaging protocol (Figure 6). The same message circle is used also between the agents.

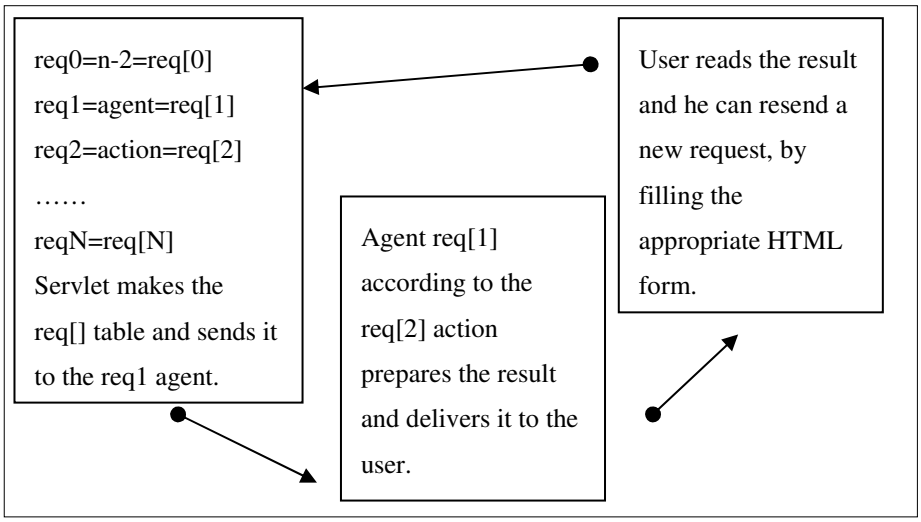

Figure 6. Message Cycle

3.5 System Database

After analysing the needs of a complete collaborating system we also found out some basic information that would help the collaboration in general. This information is decided to be stored in the System Database. The E-R model of this database is simple enough and it also stores information that can be used for security purposes (Figure 7).

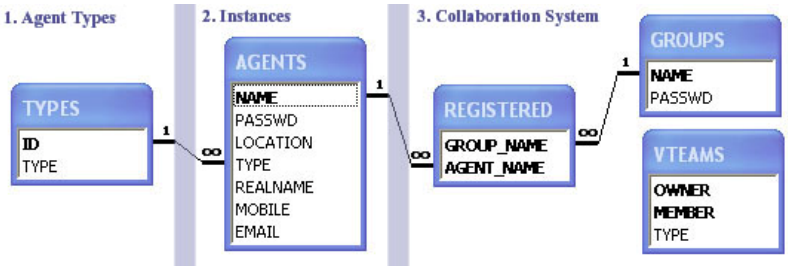

Figure 7. E-R of System Database

Initially, in the first phase of the system (creation) the table TYPES is populated with all the new agent types. As mentioned before the agent instances can be created in both system phases, where the table AGENTS is populated. Finally, while all team members are collaborating harmonically, all the other database tables are populated by creating groups and virtual teams.

3.6 Compilation and Execution of new Mobile Agents

DY.M.A.CO.S. system in order to create new types of agents it has to create the .java file that represent the new agent type, then compile them to .class files and after that it can create instances of this agent type. The problem of the compilation and the new agent invocation is overcome by the user of the Exec Class. Exec Class can execute a program with 3 ways:

1. **exec:** Executes a program without waiting for the result. This method is useful for printing a document.

2. **execWait:** Like the above method but it is waiting for the command to finish. This method is useful if someone want to execute 2 programs and the second one have to process the result of the first. For example, in our case, the command javac is followed by java.
3. **execPrint:** Executes a program and returns it is output. It is used for a command like "ls" in UNIX.

In other words, the M.AG. compiles and invoke the new agents using the above methods as follows:

1. Exec.execWait("/usr/local/bin/javac Foo.java");
2. Exec.execWait("/usr/local/bin/java Foo");

3.7 Other Technologies

The main system's idea is to create a collaboration system very fast and easy and to be able for the developers to extend these agents with more functions that support their application. For example if we have a collaboration system for an insurance company, we may also want the agents to have access to the company's database. This option is also included in the system. The developer can add some predetermined methods that support his application and these methods can be added by the users on demand, while they are creating their agent types. These predetermined methods are stored in a file (readyMethods.dat) by the developer with all other custom java libraries.

4 CASE STUDY

As mentioned above, one successful collaboration system is DITIS (Pitsillides A. et al., 2003, Samaras G. et al. 2005). We tried to implement as many functions of DITIS as we can with DY.M.A.CO.S., just to measure the time and effort that we will be consumed. We will focus on the communication part and some of the basic operations of DITIS.

In the existed phase, all DITIS actors have the same interface and they only use messages for communication. The basic operations are:

- **Next Appointment:** The system returns details about user's next appointment.
- **Today's Appointment:** From here the user can view all his appointments.
- **List Patients:** From this menu the user can find an existing patient.
- **New Patient:** From this menu the user can add a new patient to the system.
- **Equipments:** The system returns all the available equipments.
- **Patient Issues:** From this menu the user can access patient's medical details.
- **Drug Info:** The user can find any drug information from this menu.
- **Messages:** From this menu the user can read his messages.
- **Last Request:** The system returns the last user's result.
- **Statistics:** The system returns some statistics for the user.

All operations that have nothing to do with collaboration (the application specific ones) are specially formed and inserted to the predetermined method file.

Main DITIS actors are the Nurse, the Oncologist, the Physiotherapist and the Psychologist. With the use of DY.M.A.CO.S. we will try to make a collaboration system that can support these DITIS users. At first, we have to create the agent types and then set some users. As we mentioned previously, we can do both actions simultaneously.

Initially, using the login screen (Figure 8) we choose the option "Create". We also insert general user information like username, password, name, telephone and email for the creation of the agent instance along with the agent type.

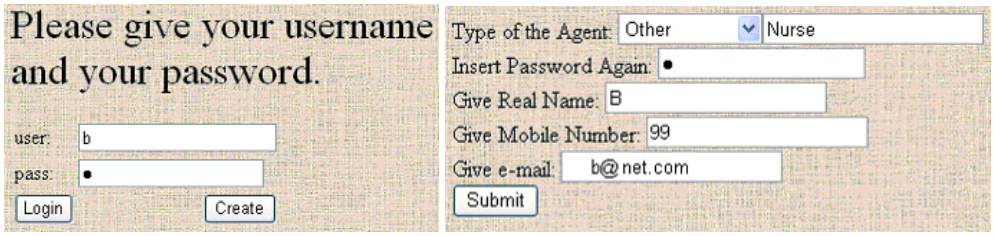

Figure 8. Create New Agent Type and Instance

After this step, we choose the collaboration capabilities that will support this agent type (Figure 9). We can choose capabilities for messaging, message groups and virtual teams. In this menu there are also some other options for calling simple SQL queries for setting up functions like the option of changing the password.

In our case study, we need to implement more functions than the default ones. We can do that selecting the "Methods" button (Figure 9). As we mention above, we add predetermined methods in order to intergrade DY.M.A.CO.S. system with any other existing system. We can select any of these methods during the creation of the agent type. Through a menu we can select some of the predetermined methods or even write our own extra SQL declarations.

Figure 9. Collaboration Options

After adding all the application specific methods, the agent is created by the M.AG. and the instance that we create simultaneously is invoked. The main menu is returned to the user for immediate usage (Figure 10). We can notice that the menu is formulated exactly by the options that we selected through these steps.

Figure 10. Final Nurse's Menu

Following the same procedural steps, we create the oncologist, the physiotherapist and the psychologist. Their menus are different according the selections that we set (Figure 11). Each actor now has his own interface according to his needs and the agents have the knowledge to communicate with each other, supporting this collaboration schema.

Figure 11. Oncologist's, Physiotherapist's and Psychologist's Menu

5 CONCLUSION AND DISCUSSION

The main limitation of DITIS system is that there was only one interface for actors. No matter the role type, all users had the same look and feel providing consequently the same functionality and services. This was also a major limitation for the collaboration because these user types have different needs. Additionally, the collaboration was supported mainly by the utilization of simple messaging exchange.

With the help of DY.M.A.CO.S. system we managed to create over the 70% of the system in about 10 minutes. Through the use of our system, we have created all the agents for all user types and additionally we provided more collaboration abilities and all needed interfaces for each user type. Finally, the only pending issue for the developer to insert the application specific methods that will support his system.

The novel implementation of DITIS that spawned from DY.M.A.CO.S. system is a much more advanced system than the previous one. As a result, this novel implementation, that was created b DY.M.A.CO.S. system, was the basis for the new phase of DITI. It is enhanced with new methods and is already given to the users for preliminary usage. The first intercourse with the new version of DITIS

showed that the DY.M.A.CO.S. is successful in creating a collaboration system fast and almost effortless.

References

Aglets Workbench, by IBM Research Group, Web Site: <http://www.trl.ibm.co.jp/aglets>

Anuff E. and Hotwired O., “Java Sourcebook”, Wiley Computer Publishing, Inc., New York, 1996.

Chess D., Grosof B., Harrison C., Levine D., Parris C. and Tsudik G., Itinerant Agents for Mobile Computing, Journal IEEE Personal Communications, Vol. 2, No. 5, October, 1995.

Concordia, by Mitsubishi Electric, Web Site: <http://www.meitca.com/hsl/Projects/Concordia>

Dikaiakos M., Kyriakou M., Samaras G., "Benchmarking Mobile-Agent Systems," Technical Report TR-2001-2a, Department of Computer Science, University of Cyprus, November 2001

Grasshopper agent platform, by GMD FOCUS and IKV++, Web Site: <http://www.grasshopper.de/>

Harrison C., Chess D. and Kershenbaum A., “Mobile Agents: Are they a good idea?”, Research Report, IBM Research Division., 1995

Lange D. and Chang D., “Programming Mobile Agents in Java”, White Paper, IBM Aglets Workbench., 1996

Lange D.and Oshima M., “Programming and Deploying Java Mobile Agents with Aglets”, Addison Wesley Longman, Inc., Reading, Massachusetts, 1998.

Mitsubishi Electric ITA, “Mobile Agent Computing”, Horizon Systems Laboratory White Paper. 1998

Pitsillides A., Pitsillides B., Samaras G., Dikaiakos M., Christodoulou E., Andreou P., and Georgiadis D., A Collaborative Virtual Medical Team for Home Healthcare of Cancer Patients, Book Chapter, M-Health: Emerging Mobile Health Systems, (R. H. Istepanian, S. Laxminarayan, C. S. Pattichis, Editors), KLUWER ACADEMIC/PLENUM PUBLISHERS, pp. 247-266, 2005.

Pitsillides B., Pitsillides A., Samaras G., Georgiadis D., Andreou P., Panteli N., DITIS: A collaborative system to support home-care by a virtual multidisciplinary team, 8th Congress of the European Association for Palliative Care, The Hague, Netherlands, April 2003.

Samaras, G., Pitsillides A. , "Client/Intercept: a Computational Model for Wireless Environments", Proc. 4th International Conference on Telecommunications (ICT'97), Melbourne, Australia, April 1997.

Samaras G., Georghiadis D., and Pitsillides A., Computational and Wireless Modeling for Collaborative Virtual Medical Teams, Book Chapter, M-Health: Emerging Mobile Health Systems, (R. H. Istepanian, S. Laxminarayan, C. S. Pattichis, Editors), KLUWER ACADEMIC/PLENUM PUBLISHERS, pp. 107-132, 2005.

Voyager, by Object Space, Web Site: <http://www.objectspace.com>

White J. E., Mobile Agents, General Magic White Paper, www.genmagic.com/agents, 1996.