ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

Agent Based Expert System for Online Assessment in Distributed Database Environment: Prototype Design and Implementation

Khurshid Alam Borbora[1], Ridip Dev Choudhury[2], Shikhar Kr. Sarma[3]

Assistant Professor, Department of Computer Science), Gauhati University Institute of Distance and Open Learning, Guwahati, India[1]

Assistant Professor, Department of Computer Science), Gauhati University Institute of Distance and Open Learning, Guwahati, India [2]

Professor & HOD, Department of IT, Gauhati University, Guwahati, India[3]

ABSTRACT: The Distance Learning scenario has been changed with the advancement of Internet technology. They key issue in Distance Learning is the assessment of students' activities. Mobile Agent approach can improve the development of Expert System Applications. In this work, we present the design and implementation of a FIPA compliant Mobile Agent based Expert System in Distributed Database Environment using JESS in the JADE framework with JIPMS support and it is deployed in J2EE platform using MVC paradigm.

Keywords: Mobile agent, expert system, JESS, online assessment, FIPA, J2EE, MVC, JADE, JIPMS, distributed database environment

I. INTRODUCTION

Over the last decade, mobile agent paradigm and technology [1],[2],[3] have shown high potential and flexibility for the design and implementation of general-purpose distributed applications, particularly in dynamic environments such as those based on the Internet. A mobile agent can be defined as an autonomous software entity which can migrate from one host to another to fulfil a goal-oriented task. Mobile agents are supported and managed by distributed software platforms known as Mobile Agent Systems (MASs) [4].

Although plenty of MASs are currently available and several critical advantages deriving from the exploitation of mobile agents have clearly been demonstrated by both academic and industrial research efforts [1],[3],[5], the following major drawbacks still hinder the widespread use of mobile agents: the lack of appropriate security mechanisms, the lack of interoperability, and the lack of software development processes and modelling languages tailored for mobile agents [4].

This paper actually represents the design and implementation of mobile agent and local agent; the mobile agent is used to move server to server and gathers question set. The servers may not have the same DBMS running on them, so therefore a local agent is developed for the different DBMS as a mediator between the mobile agent and DBMS.

II. FIPA SPECIFICATIONS

FIPA97 specifications [6] which is the first document of FIPA and it specifies the normative rules that allow agents to inter-operate, those are effectively exist, operate and can be managed.

A. *AGENT PLATFORM:*

The standard model of an agent platform, as defined by FIPA, is represented in the following Fig. 1. [6]

The Agent Management System (AMS) is the agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. Each agent must register with an AMS in order to get a valid AID. [7]

The Directory Facilitator (DF) is the agent who provides the default yellow page service in the platform. The Message Transport System, also called Agent Communication Channel (ACC), is the software component controlling all the exchange of messages within the platform, including messages to/from remote platforms. [7]

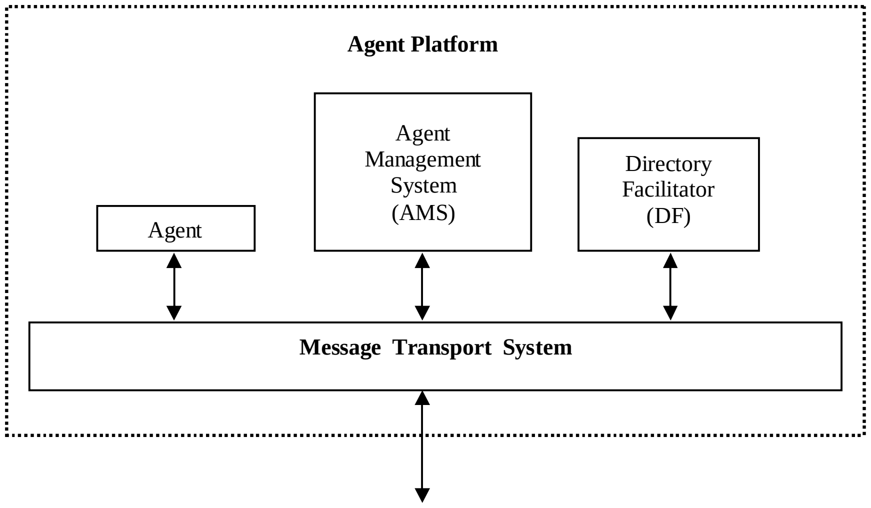

Fig. 1 – Reference architecture of a FIPA Agent Platform

There is no restriction for the technology used for the platform implementation but all should be FIPA compliant implementations. The FIPA standard also specifies the Agent Communication Language (ACL). Here, the communication between agents is based on message passing, i.e. agents communicate by formulating message and then sending formulated message to each other.

III. MOBILE AGENT DESIGN PATTERNS

Though in different case studies ([8],[9],[10] and [11]), the mobile agent technology has been successfully applied but the restrictions in its application in practical scenarios still exists. Different design patterns on mobile agents have been proposed in [12] and [13]. Below are the brief descriptions on various design patterns:

A. *Itinerary*

This design pattern is proposed in [13]. Here the agent receives an itinerary on the source agency which indicates the sequence of agencies/servers it should visit. In each of the agencies/servers, the agent executes its task locally and then continues on its itinerary. After visiting the last agency/server, the agent then goes back to its source agency/server. This design pattern is depicted in Fig. 2.

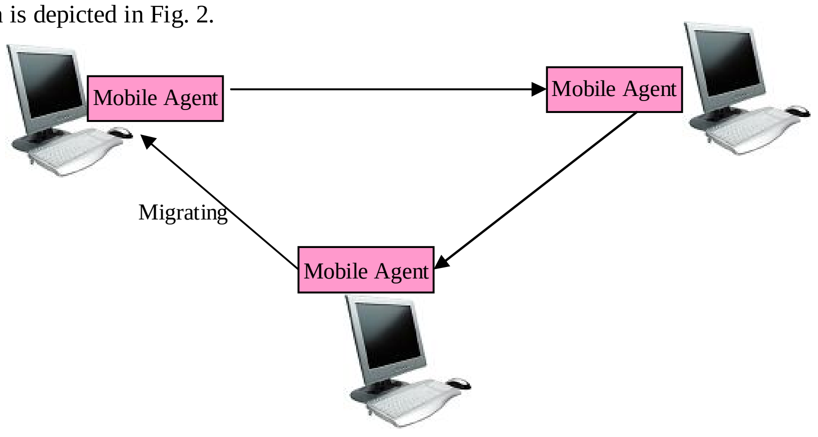

Fig. 2: Itinerary Design Pattern

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

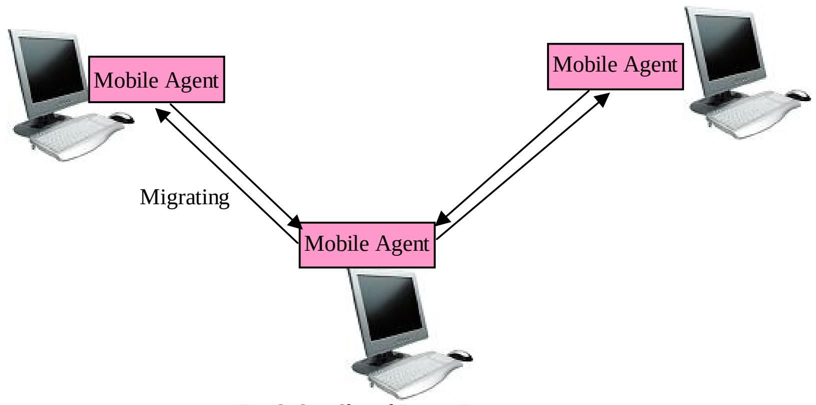

Fig. 3: Star-Shaped Design Pattern

B. *Star-Shaped*

In Star-Shaped design pattern [13], the agent has a list of agencies/servers to migrate. The agent then migrates to the first destination agency/server, executes the task, going back to the source agency/server. The agent repeats this until it visits the last agency/server in its list. Fig. 3 shows this pattern.

C. *Branching*

In the Branching design pattern [13], the agent receives a list of agencies/servers to visit. It then clones itself according to the numbers of agencies/servers in the itinerary. Each of the clones has to execute its corresponding task in an agency/server respectively. And the clones notify the source agency/server when the respective task of each of them is completed. Here, in this pattern, parallel execution of the task/tasks can be achieved. This pattern is shown at Fig. 4.

D. *Master-Slave*

Here in this Master-Slave [14] design pattern, there are two types of agents, the master agent and the other is the slave agent. The master agent gives a task to be done in a given agency/server to a slave agent. The slave agent then visits the indicated agency/server to accomplish its task and then returns with the results to the source agency/server. The master agent receives the results from the slave agent and then the slave destroys himself.

E. *MoProxy*

This design pattern is proposed in [15], proposed when an agent needs a resource, it requests to the Resource Granter, indicating the desired permissions. Then, the resource granter returns a mobile proxy for the agent in order to access the resource with the desired permissions depending on the restrictions of the resource.

F. *Meeting*

The Meeting pattern [14] suggests a way to promote local interactions between agents distributed on the net. Such interactions make possible the execution of given tasks, as well as the optimization of results. Then, the Meeting Agent, who will meet others, has a Meeting object that keeps the place and the identification of the meeting. In this way, the Meeting Agent requests the Meeting object the place of the meeting and then migrates to it. A Meeting Manager, which is an entity that manages the meeting, is responsible for notifying the agents located in the meeting place about the arrivals and exits of new ones. The Meeting object is responsible for inter-mediating the register of the agent on the manager.

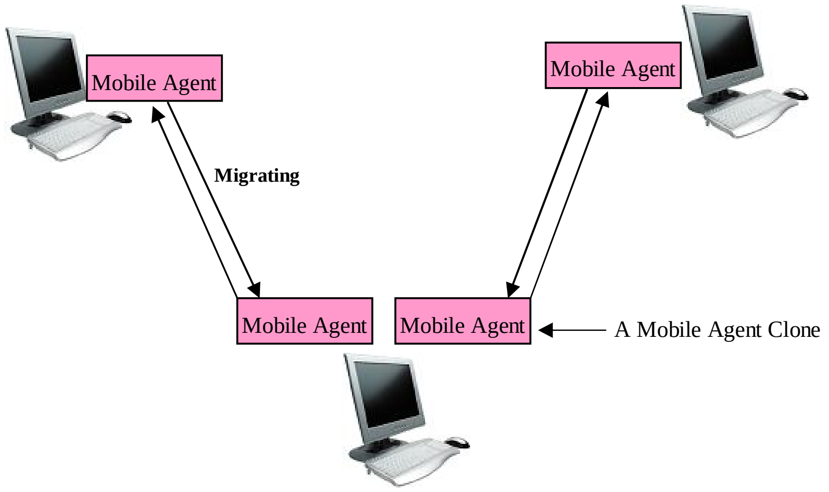

Fig. 4: Branching Design Pattern

G. *Facilitator*

The Facilitator pattern [14] defines a service that provides a name service and localization of agents with specific abilities, thus facilitating the localization of a given agent.

H. *Mutual Itinerary Recording*

This pattern [16] is a general schema that guarantees the itinerary of a given agent will be registered and tracked by other cooperative agent and vice-versa, in a disposal of mutual support. When an agent is moving between platforms, it carries the information from the last platform, the current and the next ones to the cooperative agent through an authenticated channel. The agent keeps the register of the itinerary and it always compares the itinerary that it possesses with the received one. When an inconsistence is detected it should be treated. For instance, it would either disallow the agent to visit the platform that caused the inconsistency, or suspend the functioning of an agent, or send the agent back to the source agency.

IV. MOBILE AGENT DESIGN PATTERN USED IN EESOA

In the design and implementation of the EESOA [17] framework, we have considered the Itinerary design pattern for our mobile agent, QSGMA. In today's computing world, the speed of the networks: LAN, MAN, WAN, Internet etc., is increasing day-by-day. Therefore, we have given priority to the load on the server where the EESOA resides.

We have designed a mobile agent which will start with the list of servers to be visited and then visit each of the servers one-by-one and gather questions and at last goes back to the source server. We also have designed local agent, QSGLA which must be in every server, which is a communicating agent with the DBMS in each of the servers, Fig. 5.

V. SOFTWARE TOOLS USED

This work is an extension of ESOA, a Web based Expert System for Online Assessment, which has implemented using:

- JESS (7.1p2): JESS is a Java based Rule Engine and scripting environment and it also supports Java API.[18]
- J2EE Platform: It is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications. [19]
- J2SDK (1.5.0): The Java 2 Software Development Kit is the package containing the java compiler and other tools needed to convert your java programs into executable class files which can be run by the java interpreter. The J2SDK can be downloaded from java.sun.com for free. Versions are available for windows, linux and solaris and Mac. [20]

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

- MySQL Server (Version 5.5): It is the world's most widely used open source relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases. [21]
- Apache HTTP Server (Version 2.2): The Apache HTTP Server commonly referred to as Apache, is a web server software program notable for playing a key role in the initial growth of the World Wide Web. It became the first web server software to surpass the 100 million website milestone. [22]
- Apache Tomcat (Version 6.0): Apache Tomcat (or simply Tomcat, formerly also *Jakarta Tomcat*) is an open source web server and servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run. [23]

Apart from the above software tools, the following software tools are used in the EESOA implementation:

- JADE (Version 4.3.0)
- JIPMS (Version 1.2)

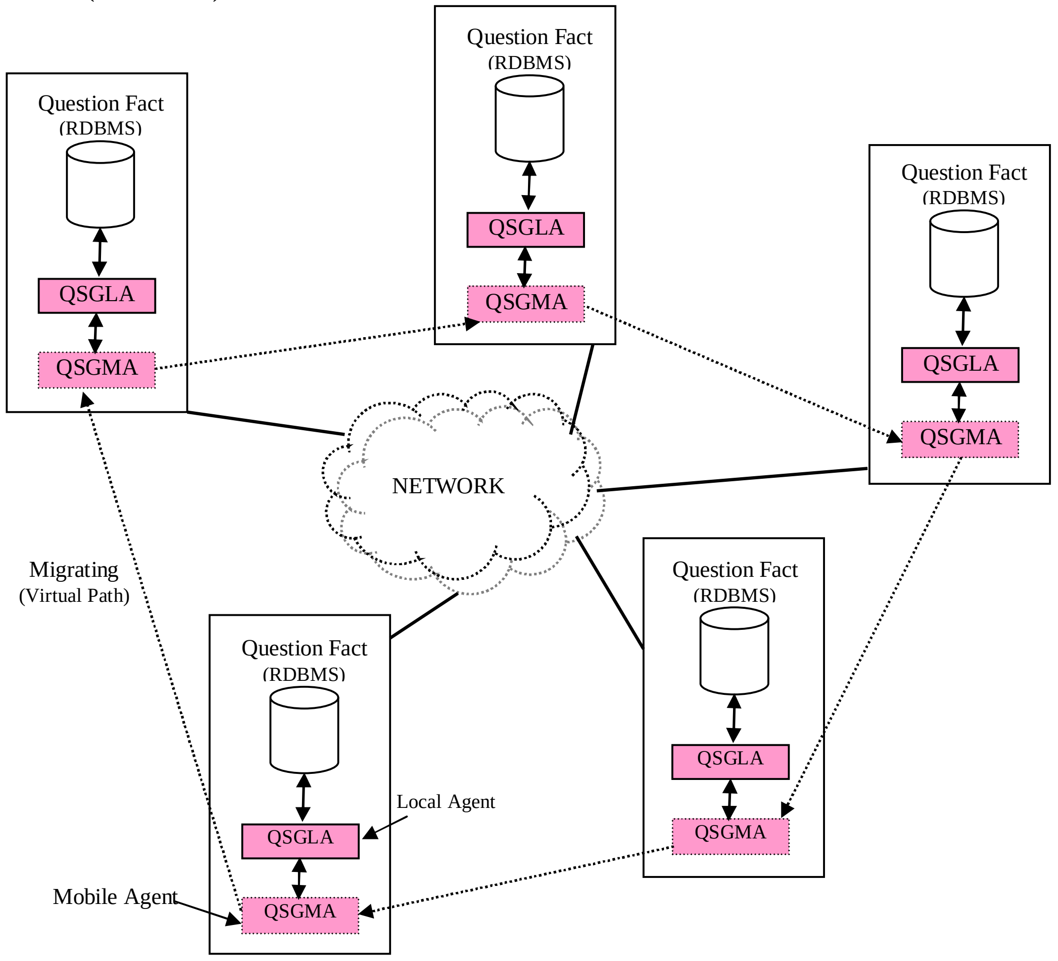

Fig. 5: Design Pattern used in EESOA

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

A. *JADE and JIPMS*

JADE (Java Agent Development Framework) is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents. It includes two main products: a FIPA-compliant agent platform and a package to develop Java agents. JADE has been fully coded in Java and an agent programmer, in order to exploit the framework. It is written in Java language and is made of various Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application dependent tasks. [7]

1) Components of JADE: JADE is a middleware that facilitates the development of multi-agent systems. It includes [24]:

- A runtime environment where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host.
- A library of classes that programmers have to/can use (directly or by specializing them) to develop their agents.
- A suite of graphical tools that allows administrating and monitoring the activity of running agents.

When the JADE platform is launched, both the AMS and the DF (discussed in section 2) are automatically created. Currently, the following tools are available [7]:

- Remote Management Agent, RMA for short, acting as a graphical console for platform management and control. Moreover, the RMA console is able to start other JADE tools.
- The Dummy Agent is a monitoring and debugging tool, made of a graphical user interface and an underlying JADE agent. Using the GUI it is possible to compose ACL messages and send them to other agents.
- The Sniffer is an agent that can intercept ACL messages while they are in flight, and displays them graphically using a notation similar to UML sequence diagrams. It is useful for debugging your agent societies by observing how they exchange ACL messages.
- The Introspector is an agent that allows to monitoring the life cycle of an agent, its exchanged ACL messages and the behaviours in execution.
- The DF GUI is a complete graphical user interface that is used by the default Directory Facilitator (DF) of JADE and that can also be used by every other DF that the user might need. In such a way, the user might create a complex network of domains and sub-domains of yellow pages.
- The LogManagerAgent is an agent that allows setting at runtime logging information, such as the log level, for both JADE and application specific classes that use Java Logging.
- The SocketProxyAgent is a simple agent, acting as a bidirectional gateway between a JADE platform and an ordinary TCP/IP connection. ACL messages, travelling over JADE proprietary transport service, are converted to simple ASCII strings and sent over a socket connection. Viceversa, ACL messages can be tunnelled via this TCP/IP connection into the JADE platform. This agent is useful, e.g. to handle network firewalls or to provide platform interactions with Java applets within a web browser.

2) Containers and Platforms: Each running instance of the JADE runtime environment is called a Container as it can contain several agents. The set of active containers is called a Platform. A single special Main container must always be active in a platform and all other containers register with it as soon as they start. It follows that the first container to start in a platform must be a main container while all other containers must be "normal" (i.e. non-main) containers and must "be told" where to find (host and port) their main container (i.e. the main container to register with). [24]

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

3) *JIPMS:* JIPMS (JADE Inter-platform Mobility Service) is a service which allows Inter-platform mobility support to JADE. It is an extra module, which does not come with JADE package and can be downloadable from the net. [17]

VI. Implementation of Agents

A. *Implementation of the Mobile Agent (QSGMA)*

The agent, QSGMA, is the mobile agent and which is running in all the question database servers. This agent is inherited from the built-in class in JADE, jade.core.Agent [7,24].

```
import jade.core.Agent;

public class QSGMA extends Agent {

        // Member-Variable Declarations
          Question[] qarray;
         int totalqhosts;
         String[] qhostips;
         boolean state;
         int curhostno;
        ……………………………
        ……………………………

        // Code the constructors and different methods
        .........................................
        ……………………………
        protected void setup() {
              …………………………
              …………………………
        }
}
```

One of the member-variables, qarray[], which is an array of type Question, is used to store question obejects after fetching from the servers. Question is a simple class:

```
public class Question {
            String qid;
            String question;
            String qhostip;
            …………………………
            ………………………....
}
```

The member, totalqhosts, holds the total no of servers and the String array, qhostips, will have the ip-addresses of the servers those are to be visited by the mobile agent.

When the agent QSGMA is executed, at first the constructor is executed, where the members- totalqhosts and qhostips[], are initialized, and the Agent is given an identifier and is registered in the AMS, and is put in the ACTIVE state

identifier=localname+"@"+platformmachine:port+"/JADE"

The setup() method, is intended to include agent initializations, and is used to:

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

- (optionally) modify the data registered with the AMS
- (optionally) attribute services to the agent, and register with one or more domains (DFs)
- add behaviours (tasks) which will be scheduled as soon as the setup() method returns

```
protected void doDelete() {
   ………………………….
   ………………………….
   }
```

The doDelete() method is used to stop the agent execution.

```
protected void takeDown() {
   ………………………….
   ………………………….
}
```

The takeDown() method is used for any cleanup (e.g. unregister from DF) before the agent is destroyed.
The actual job an agent has to do is typically carried out within "behaviours". A behaviour represents a task that an agent can carry out. In our case the question fetch task is implemented as an object of the class, QFetchBehaviour, that extends jade.core.behaviours.SimpleBehaviour.

```
public class QFetchBehaviour extends SimpleBehaviour {
  // Data-member Declarations & Constructor Definition
  Agent agent;
  ……………………….
  ……………………….
  public void action() {
          if( state == TRUE ){
            …………………….
            agent.doMove ( destination-machine-location );
            ……………………
            state = FALSE;
          }
          else
          {
                    ACLMessage mmessage = new  ACLMessage ( ACLMessage.REQUEST );
                    …………………….
                    …………………….
                    send ( mmessage );
                    block();
          }
}

public boolean done() {
  …………………….
  …………………….
  }
}
```

The action() method actually defines the operations to be performed when the behaviour is in execution and the done() method (returns a boolean value), that specifies whether or not a behaviour has completed and have to be removed from the pool of behaviours an agent is carrying out. [7]

doMove() method, takes a parameter of type jade.core.Location, migrates the agent to a new location identified by the parameter.

The ACLMessage class is used for communication between the agents which is basically asynchronous message passing. The performative can be REQUEST, if the sender wants the receiver to perform an action. So, inside the QSGMA agent class, there is a member-variable:

```
QFetchBehaviour hBehaviour;
```

Inside the setup() method, the statement,

```
hBehaviour =  new QFetchBehaviour(course, subject, topic,  paramvalue);
addBehaviour(hBehaviour);
```

The addBehaviour() method adds the behaviour, QFetchBehaviour, to the agent for the tasks to be executed. The parameters:

(a) course: indicates the course for the assessment,
(b) subject: indicates the subject for the assessment,
(c) topic: indicates the topic of the assessment,
(d) paramvalue: is of String type and it holds the values, to be treated as percentage of questions to be supplied according to toughness of the questions, separated by "|".

The mobile agent, QSGMA, will communicate with QSGLA for retrieval of question from the respective server. The QSGLA will then communicate with the DBMS and get the result questions and give them to QSGMA. QSGMA then update his question data and then move to the next server and so on until it visits the last server.

B. *Implementation of the Local Agent (QSGLA)*

The local agent, QSGLA, is designed in such a way that it can communicate with most of the DBMS for the retrieval of questions.

The implementation of the local agent, QSGLA, is given below:

```
public class QSGLA extends Agent {
        // Member-Variable Declarations
        static int no_of_agents;
        Question[] qarray;
       ..................................
        ………………………………

        // Code the constructors and different methods
        .........................................
        ………………………………
}
```

One of the member-variables, qarray[], which is an array of type Question, is used to store question objects after fetching from the servers. Question is a simple class, whose definition is already mentioned above.

```
protected void setup() {
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

```
        ………………………….
        ………………………….
}
```

The setup() method is intended to include agent initializations. In the setup() method, the method doLisen() is called which will listen for the requests from agent QSGLA.

```
protected void doDelete() {
        ………………………….
        ………………………….
}
```

The doDelete() method is used to stop the agent execution. The takeDown() method is used for any cleanup (e.g. unregister from DF) before the agent is destroyed.

```
protected void takeDown() {
        ………………………….
        ………………………….
}
```

The actual job an agent has to do is typically carried out within "behaviours". A behaviour represents a task that an agent can carry out. In our case the question fetch task is implemented as an object of the class, QRetBehaviour, that extends jade.core.behaviours.SimpleBehaviour.

```
public class QRetBehaviour extends SimpleBehaviour {
            //Code of the Constructors
            ……………………….
            ……………………….

            //Code of the different methods
            public void action() {
                        ACLMessage  msg = myAgent.receive();
                        if (msg!= null){
                                    ………………………
                                    ………………………
                                    return true;
                        }
                        else {
                                    ………………………….
                                    ………………………….
                                    block();
                                    return false;
                        }
                        ……………………….
                        ……………………….
            }
            public boolean done() {
                        ……………………….
                        ……………………….
            }
}
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

An agent can pick up messages from its message queue by means of the receive() method [24]. The block() method marks the behaviour of the agent as "blocked". So, inside the QSGLA agent class, there is a member-variable:

QRetBehaviour hBehaviour;

Inside the setup() method, the statement,

hBehaviour = new QRetBehaviour();
addBehaviour(hBehaviour);

The addBehaviour() method adds the behaviour, QRetBehaviour, to the agent for the tasks to be executed. Inside the code, if (msg!= null){.., the job of question retrieval and sending the result set of questions back to the QSGMA.

VII. TESTING

The Expert System, EESOA, is implemented and tested in a computer laboratory LAN. Four nodes have been configured in such a way that they can be treated as database servers with local IPs. Out of the four servers, in two servers MySQL DBMS is running and in the other two, PostgreSQL is running. In one of them, the EESOA is running.

In the database server, where ESOA is running, has students' history data as well as question bank. And in the remaining three database servers, only the question banks exist. In all the database servers, the JRE 1.5.0 is installed. Also JADE 4.3.0 and JIPMS 1.2 are installed in all the servers. And all the remaining nodes are treated as clients. The system is then tested with a small group of students. The assessment has been done a no. of times and the system performs properly.

VIII. CONCLUSION AND FUTURE WORK

This paper presents the prototype design and development of a mobile agent based Expert System for Online Assessment (EESOA) in distributed database environment for the students and more specifically for the ODL learners. The prototype of the system is designed, implemented and then tested in a small LAN. Since the environment, where we tested the system, is a computer laboratory LAN where no real-time traffic is there. The system is to be tested in a real-time environment and then we can actually measure the performance of the system.

The students' history will increase with the enrolment of new students and also with the assessment done by the students. Here this case the students' history will be a huge data-bank. In later time, this rapid increase in the students' history data will affect the performance of the system. So, for the job of fetching students' history, mining-technique can be implemented.

The rules in the system are predefined. Here we can make improvement in the generation of rules with the help of machine learning techniques.

Also the work can be extended by also distributing the student database as the way the question database was distributed here. This extension will be useful for the universities/institutions in collaboration for introduction of new courses jointly.

REFERENCES

1. Kotz, David, George Cybenko, and Daniela Rus. "Mobile agents: Motivations and state-of-the-art systems." (2000).
2. Gschwind, Thomas. "Comparing object oriented mobile agent systems." (2000).
3. Braun, Peter, and Wilhelm R. Rossak. *Mobile agents: Basic concepts, mobility models, and the tracy toolkit*. Elsevier, 2005.
4. Fortino, Giancarlo, Alfredo Garro, and Wilma Russo. "Achieving mobile agent systems interoperability through software layering." *Information and software technology* 50.4 (2008): 322-341.
5. Lange, Danny B., and Oshima Mitsuru. *Programming and Deploying Java Mobile Agents Aglets*. Addison-Wesley Longman Publishing Co., Inc., 1998.
6. Foundation for Intelligent Physical Agents. Specifications. 1997. Available from *http://www.fipa.org*
7. JADE Programmer's Guide by Fabio Bellifemine, Giovanni Caire, Tiziana Trucco (TILAB, formerly CSELT), Giovanni Rimassa (University of Parma) available from *http://jade.tilab.com/doc/programmersguide.pdf*
8. Hayzelden, Alex LG, and John Bigham, eds. *Software agents for future communication systems*. Springer, 1999.
9. Bernardes, Mauro Cesar, and Edson dos Santos Moreira. "Implementation of an intrusion detection system based on mobile agents." *Software Engineering for Parallel and Distributed Systems, 2000. Proceedings. International Symposium on.* IEEE, 2000.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 2, February 2014

10. Klein, Cornel, et al. "Extension of the Unified Modeling Language for Mobile Agents." (2001): 116-128.
11. Glitho, Roch H., Edgar Olougouna, and Samuel Pierre. "Mobile agents and their use for information retrieval: a brief overview and an elaborate case study." *Network, IEEE* 16.1 (2002): 34-41.
12. Aridor, Yariv, and Danny B. Lange. "Agent design patterns: elements of agent application design." *Proceedings of the second international conference on Autonomous agents*. ACM, 1998.
13. Yoshioka, Nobukazu, et al. "Security for mobile agents." *Agent-Oriented Software Engineering*. Springer Berlin Heidelberg, 2001.
14. Oshima, M. I. T. S. U. R. U., and DANNY B. Lange. "Programming and Deploying Java Mobile Agents with Aglets." *Canada, August* (1998).
15. A.Kr. Singh, R. Sankar, and V. Jamwal. Design patterns for mobile agent applications.
16. Jansen, Wayne, Peter Mell, Tom Karygiannis, and Donald Marks. "Mobile agents in intrusion detection and response." In *Proceedings of the 12th Annual Canadian Information Technology Security Symposium*, vol. 12. 2000.
17. Borbora, Khurshid Alam, Ridip Dev Choudhury, and Shikhar Kumar Sarma. "A FRAMEWORK FOR AGENT BASED EXPERT SYSTEM FOR ONLINE ASSESSMENT IN DISTRIBUTED DATABASE ENVIRONMENT." *International Journal* (2013).
18. Menken, Maarten. "JESS tutorial." *Vrije Universiteit, Amsterdam, The Netherlands* (2002): 1-57.
19. http://www.webopedia.com/TERM/J/J2EE.html
20. http://www.bookterra.com/index.php/Programming:Java:J2SDK
21. http://en.wikipedia.org/wiki/MySQL
22. http://en.wikipedia.org/wiki/Apache_HTTP_Server
23. http://en.wikipedia.org/wiki/Apache_Tomcat
24. JADE Tutorial: JADE Programming for Beginners by Giovanni Caire (TILAB, formerly CSELT) available from http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf