Chapter 7: System Development & Implementation of EESOA

7.1 Software Process Model Used

A software process model is a description of the sequence of activities carried out in a Software Engineering project. A Software Process Model in general represents a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution [1]. Such models can be used to develop more precise and formalized descriptions of software life cycle activities.

In the development of EESOA, I have used the process model that incorporates some elements from each of the following three models:

![Waterfall Process Model Diagram]

Figure 7.1 Waterfall Process Model
Waterfall Model: In the waterfall software phase model, where software evolution proceeds through an orderly sequence of transitions from one phase to the next in order [2]. The following figure, Figure 7.1, represents the waterfall model.

Incremental Development and Release: In case of incremental release (figure 7.2), the first step is to develop systems with basic and essential functions and release it [3]. After that more capable and stable versions of the system are to be provided at regular intervals. This approach has also been extended through the use of software prototyping tools and techniques.

Reuse-oriented Software Engineering Model: This process model is basically based on systematic reuse of some existing software components, i.e. here systems are integrated from existing components or Commercial-off-the-shelf systems. Figure 7.3 depicts the process model.
7.2 Software Tools Used

I have implemented EESOA using MVC paradigm. The following software tools are used in the design of ESOA.

- **J2EE Platform**: It is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online [4]. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multi-tiered, Web-based applications.

- **J2SDK 1.5.0**: The Java 2 Software Development Kit [5] 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, Solaris and Mac.

- **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 (WWW) [6]. It became the first web server software to surpass the 100 million website milestone.

- **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) [7]. Tomcat implements the Java Servlet and the Java Server
Chapter 7: System Development & Implementation of EESOA

Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run.

- MySQL Server Version 5.5 [8]: 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.

- PostgreSQL Server Version 8.4 [9]: It is an object-relational database management system. It has emphasis on extensibility and standards-compliance.

- mysql-connector-java-3.1.12 [10]: It is the driver for MySql-JDBC which enable the developers to build database applications by using Java language.

- Postgresql-8.4-704 [9]: It is the driver for PostgreSQL-JDBC which enables the developers to build databases applications by using the Java Language.

- JESS 7.1p2 [11]: JESS is a Java based Rule Engine and scripting environment and it also supports Java API.

- JADE Version 4.3.0 [12]: It is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents. JADE has been fully coded in Java and an agent programmer, in order to exploit the framework.
• JIPMS 1.2 [13]: 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.

7.3 Strategy in Assessment

In EESOA, for self assessment, a student has to follow the following steps [14]:

(a) Log-in to the system, using username and password.

(b) Select the course for which the self-assessment is to be performed.

(c) From the subjects list select the particular subject for which the self-assessment is required.

(d) Now, select the unit from the unit list and the assessment continues.

In the implementation of EESOA, I have considered a few rules (as in ESOA). Rules are implemented in XML file and are designed to emulate the techniques applied by the teachers/experts in a particular domain/field to assess the knowledge levels of students. Here I have divided the assessment techniques, used by a teacher in a classroom, into three categories: (a) knowledge level judgement of a student prior to the present assessment; (b) while asking questions to the student for the present assessment; (c) for coming to a conclusion after judging the student's answers for that particular assessment.
Thus considering the above categories of, I have also categorized the rules in to three types: student-history-assessment rule, question-generation rule and answer-check rule. Few rules for each of the types/categories are implemented in EESOA (as in ESOA).

Rules will be fired depending upon the student model (discussed in chapter 2). For each and every attempt by a student, the inference engine has to search in the student model for the assessment history of the students. Up to the last completed attempts, the knowledge level assessment of the student is carried out for the present attempt. And thus accordingly the questions are presented to the student for each attempt. When an assessment starts the EESOA engine gathers the student’s academic and assessment history, student-history-assessment rule(s) is/are applied to judge the merit of student.

After judging the merit of the student who takes part in the assessment, the question-generation rule(s) is/are applied to generate the questions’ pattern for fetching the questions for assessment.

When the assessment completes, the answer-check-rule(s) is/are applied for the processing the result.

### 7.4 Representation of Facts and Rules

Like ESOA, here also I have used two types of facts in the expert system:

(a) student model (student facts, discussed in chapter 2),

(b) question database (question facts).
In EESOA, the question database is distributed among the different servers having their own DBMS. But the structure of each and every database in the servers is the same.

Each of the questions in the question databases is tagged with one from the five types of toughness: Very Easy (VE), Easy (E), Moderate (M), Tough (T) and Very Tough (VT). The toughness assignment for each of the questions is determined by the peer domain expert/teacher.

In JESS, there are three ways for representing facts: (a) Ordered facts (b) Unordered facts (c) Shadow facts. In case of ordered and unordered facts, those can be created entirely by Jess by using the JESS working memory. But the shadow facts connected to Java objects and they act as "bridges" that let JESS reason about things that happen outside of working memory.

In JESS, facts can be represented by three ways: (a) Ordered facts (b) Unordered facts (c) Shadow facts.

I have also designed the student model with few modifications (as in case of ESOA) and the facts are maintained externally in a MySql database. But the question bank is distributed among different database servers, as mentioned already in the previous chapters, and so the DBMSs running in the servers are may be the same or different. So, therefore as mentioned, overall the question data bank is also maintained externally.

For integration of MySql/PostgreSQL with JESS, the JSP, Servlet and JavaBean technologies are used. And in this way the EESOA is able to load the data from databases in the form of facts into the working memory of
JESS. JavaBean technology is basically used for implementing the shadow facts. The tables, Table 7.1 and 7.2, shows only a part of student model.

Table 7.1: Student model (TEST_MASTER table)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assessment_id</td>
<td>string</td>
<td>Unique Test Id</td>
</tr>
<tr>
<td>agent_id</td>
<td>string</td>
<td>The Agent Id of the Mobile Agent</td>
</tr>
<tr>
<td>std_id</td>
<td>string</td>
<td>Unique Student Id from STUDENT_MASTER</td>
</tr>
<tr>
<td>course_id</td>
<td>string</td>
<td>Unique Course Id from COURSE_MASTER</td>
</tr>
<tr>
<td>Paper_id</td>
<td>string</td>
<td>Unique Paper Id from PAPER_MASTER</td>
</tr>
<tr>
<td>unit_id</td>
<td>string</td>
<td>Unique Chapter Id from CHAPTER_MASTER</td>
</tr>
<tr>
<td>total_ques</td>
<td>integer</td>
<td>Total No. of Questions</td>
</tr>
</tbody>
</table>

Start of qnos (a composition of the following fields)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>very_easy</td>
<td>integer</td>
<td>Total No. of Very Easy Questions</td>
</tr>
<tr>
<td>very_easy_correct</td>
<td>integer</td>
<td>Total No. of Very Easy Questions correctly answered</td>
</tr>
<tr>
<td>easy</td>
<td>integer</td>
<td>Total No. of Easy Questions</td>
</tr>
<tr>
<td>easy_correct</td>
<td>integer</td>
<td>Total No. of Easy Questions correctly answered</td>
</tr>
<tr>
<td>moderate</td>
<td>integer</td>
<td>Total No. of Moderate Questions</td>
</tr>
<tr>
<td>moderate_correct</td>
<td>integer</td>
<td>Total No. of Moderate Questions correctly answered</td>
</tr>
<tr>
<td>tough</td>
<td>integer</td>
<td>Total No. of Tough Questions</td>
</tr>
<tr>
<td>tough_correct</td>
<td>integer</td>
<td>Total No. of Tough Questions correctly answered</td>
</tr>
<tr>
<td>very_tough</td>
<td>integer</td>
<td>Total No. of Very Tough Questions</td>
</tr>
<tr>
<td>very_tough_correct</td>
<td>integer</td>
<td>Total No. of Very Tough Questions correctly answered</td>
</tr>
</tbody>
</table>

End of qnos

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_date</td>
<td>Date</td>
<td>Date of Test</td>
</tr>
<tr>
<td>test_start_time</td>
<td>date/time</td>
<td>Starting Time of Test</td>
</tr>
</tbody>
</table>
Since we are representing the facts using the shadow fact technique, therefore we create a JavaBean which represents the structure of a relation/table in the fact database. And for accumulation of facts in working memory there must be templates, generally consisting of two parts – a name and a set of slots. The slots in the templates represent the properties of the JavaBeans.

Shadow facts serve as a connection between the working memory and Java applications in which JESS is running. The JESS functions -
• *defclass* is used to create a *deftemplate* in JESS.

• The *definstance* command creates individual *shadow facts*.

In real time application, it is required that the rule engine responds to events outside JESS. JESS allows regular java objects in its working memory which are instances of classes, where the only prerequisite is that, they be JavaBeans.

There is a class Introspector in the java.beans API which examines a JavaBean and finds properties defined by the get and set methods as shown in Figure 7.4 and generates a deftemplate. [15]

The following example shows the Java class of student performance history fact JavaBean.

```java
public class Student_Per_History {
    .........................................................
    public String getAssessment_id() ......
    public void setAssessment_id(String) ......
    public String getCourse_id() ......
    public void setCourse_id(String) ......
    .........................................................
    public int getTotal_ques() ......
    public void setTotal_ques(int) ......
    .........................................................
}
```

It can also be noted that I have used some intermediate temporary tables for creating different *deftemplates* for various processing by the JESS engine.
The following example shows another template generated with the help of Introspector class:

(deftemplate Student_Assessment (slot std_id) (slot assessment_id) (slot course_id) (slot paper_id) (slot unit_id) (slot assessment_date) (slot total_ve_question) (slot correct_ve_question) (slot total_e_question) (slot correct_e_question) (slot total_m_question) (slot correct_m_question) (slot total_t_question) (slot correct_t_question) (slot total_vt_question) (slot correct_vt_question) (slot agent_id))

In EESOA, like ESOA, the rule base is also designed using JessML. JessML is actually XML-based rule language provided by JESS. It is hybrid imperative/declarative language.

A rule contains a name element, a lhs element, and a rhs element. The rhs element contains a collection of funcall elements representing the rule's
actions. The Ihs element contains groups and patterns. The JessML rules are translated into the JESS rule using XSLT.

For example, following is an example of student-history-assessment rule:

\[(\text{defrule FetchStdAcHistory})\]
\[
\text{?acdh<- (Student\_AC\_History (std\_id \?sid))} \Rightarrow 
\text{(calculate\_ac\_performance \?acdh)} \]

\[(\text{defrule FetchStdPerHistory})\]
\[
\text{?pph<- (Student\_Per\_History (std\_id \?sid) (course\_id \?cid) (paper\_id \?pid) (unit\_id \?unitid))} \Rightarrow 
\text{(calculate\_assessment\_performance \?pph)} \]

The above two rules are triggered when a student logs-in to EESOA for assessment, may be for self-assessment and conducted assessment. At first the rule, FetchStdAcHistory, is triggered and the student’s academic history is loaded into the JESS working memory. And then function, “calculate_ac_performance” is called when the retrieval completed.

Next, FetchStdPerHistory rule is triggered, which loads the student’s performance history into the working memory of JESS. After loading, now, the “calculate_assessment_performance” function is called.
Apart from these two rules, I have also defined some more student history assessment, question generation and assessment result processing rules.

There are similarities between the JESS working memory and a database. JESS working memory is filled with indexed and structured data and generally it is being accessed by pattern matching from rule(s).

There are two methods of querying the working memory: (a) The linear Search and (b) the `defquery` construct

Since the Linear Search method is not very practical for a large working memory size, therefore I have used the `defquery` construct for query the working memory.

### 7.5 The Agent Architecture Components of EESOA

In case of the agent architecture in EESOA, I have used two different entities: mobile agent (QSGMA) and local agent (QSGLA). We will detail components of each entity in what follows (Figure 7.5):
Figure 7.5: The Agent Architecture Components of EESOA
7.5.1 Mobile Agent (QSGMA) and Local Agent (QSGLA)

- **Mobile Agent (QSGMA)**

Mobile agent contains code, static and dynamic data and modules: Question Set Pattern Generation Module, Distributed Environment List Set Module, QSGMA Query Module, QSGMA Receive Module and QSGMA Migration Module with some libraries.

- **Code**

Created by the owner, it expresses mobile agent knowledge, and constituted by above mentioned modules.

- **Static data**

Static data are basically unchangeable data such as creator identity, data used during mobile agent's execution and mobile agent itinerary.

- **Dynamic data**

This part basically contains the question set fetched from each and every database server in the distributed database environment. This part is dynamic because it evolves in every migration.

- **And different modules, which are discussed in next sub-sections in detail.**
Local Agent (QSGLA)

Like the mobile agent, the local agent also contains code, static and dynamic data and QSGLA Receive & Query Module with some libraries.

- **Code**

  Created by the owner, it expresses local agent knowledge, and constituted by above mentioned module.

- **Static data**

  Static data are basically unchangeable data such as creator identity, data used during local agent’s execution.

- **Dynamic data**

  This part basically contains the question set fetched from a particular database server in the distributed database environment for which the particular local agent is working.

  - And different modules, which are discussed in next sub-sections in detail.
7.5.2 Question Set Pattern Generation Module

This module is used by the EESOA, for generation of question pattern for a particular student attempting a self-assessment or conducted assessment. The flow of this module is shown in Figure 7.6.

The working principle of this module is described below:

- On receiving the student's knowledge level (who takes the assessment process), processed by the EESOA engine by firing the rules for student model, the EESOA engine again fetches rules question-generation-rules and calculates the pattern for the question set to be given for assessment to the student.
The module then take the TEST_MASTER relation/table, which contain the information of each and every assessment taken-up by each of the students, and insert a row with the values for:

- test_id
- std_id
- course_id
- paper_id
- unit_id
- total_ques
- toughness_levels
  - very_easy
  - easy
  - moderate
  - tough
  - very_tough
- test_date
- test_initialization_time
- test_start_time
and remaining field values left with no value assigned.

- The module then prepares a parameter list with values for:
  - Test Id
  - Course Code
  - Subject Code
  - Unit Code
  - Very Easy Question Nos
  - Easy Question Nos
  - Moderate Question Nos
  - Tough Question Nos
  - Very Tough Question Nos

And the module then interacts with the in-built JADE Platform Communication Module, which is responsible for AMS and DF of the JADE Platform, with the above parameter list.
7.5.3 Distributed Environment List Set Module

This module generates a parameter list containing the information for the distributed database environment. This module works in concurrence with the Question Set Pattern Generation Module. Figure 7.7 depicts the workflow of this module. The working principle of this module is as follows:

- This module fetched distributed environment information from the DDE_MASTER relation.

- The fetched information is stored in internal data structure whose structure is:
  
  - test_id
  
  - host_ip
  
  - ping_date_time
  
  - active_y_n
• Each and every host is then tested whether the host is reachable or not and this is carried out by pinging each host.
  
  o For each and every pinged result the internal data structure is updated for the value of the data-member “active_y_n”. If the particular host is found reachable then the value for the “active_y_n” member is set to “1” otherwise set to “0”.

• For each of the updated entry in the internal data-structure, new entries are made to the relation “DDE_TRANS” for the particular Test Id. And the relation “DDE_MASTER” is updated with the necessary values for the necessary fields.
• The module then prepares a parameter list with values for:
  
  o Test Id

  o Total Hosts

  o Host Ips with Local Agents

    • Host Ip

    • Host Name

    • OS Type

    • DBMS System

    • DBMS Version

    • Local Agent

• It then interacts with the in-built JADE Platform Communication Module, which is responsible for AMS and DF of the JADE Platform, with the above parameter list.
7.5.4 QSGMA Query Module

This module is responsible for the initiation and activation of the mobile agent, termed as QSGMA (Question Set Generation Mobile Agent). This module has few important responsibilities, which includes the formulation of travel decision, question fetching formulation and interacting with the local agent, termed as QSGLA (Question Set Generation Local Agent), for fetching the questions. The following figure (Fig. 7.8) shows the work-flow of the module. The JADE Platform Communication Module, after receiving the parameters regarding the question set pattern information and distributed database environment, invokes this module.

The working principle of this module is as follows:
• This module creates a mobile agent.

• Based on the values passed in the parameter list, this module also creates three internal structures.

• The QFP Structure is initialized with the values received from the parameters. The structure of it is:
  
  o test_id
  
  o total_questions
  
  o question_pattern
    
    • very_easy_nos
    
    • easy_nos
    
    • moderate_nos
    
    • tough_nos
    
    • very_tough_nos
  
• The DDE_TDH Structure is initialized with the values received and calculated from the parameters. The structure of it is:
  
  o host_ip
  
  o total_questions
  
  o visit_count
The module also initializes another internal structure, termed as QF, for storing the questions fetched from the distributed database hosts. The structure of QF is:

- agent_id
- host_ip
- question_id
- The module now updates the relation TEST_MASTER, which contains the information about all the assessment tests taken-up by the students or conducted (the structure of the TEST_MASTER is already mentioned in the Question Set Pattern Generation Module), for the current assessment for which this process is carrying out. The fields/attributes' values be updated for this particular Test Id are:
  - test_date
  - test_initialization_time
  - test_start_time
  - test_end_time
  - agent_id
• This module also sets few other instance variables of the mobile agent, which are:
  
  o **eurhostno** (for storing the current host no)
  
  o **totalqhosts** (for total hosts to be visited)
  
  o **state** (for current state of the agent: active/suspended) etc.

• The module now prepares the parameter list for the local agent for actual question fetching from the local database which is a part of the distributed database environment. The parameters in the parameter list are:
  
  o **Agent Id**
  
  o **Course Code**
  
  o **Subject Code**
  
  o **Unit Code**
  
  o **Question Fetch Nos**
    
    ▪ **Very Easy**
    
    ▪ **Easy**
    
    ▪ **Moderate**
    
    ▪ **Tough**
    
    ▪ **Very Tough**
Chapter 7: System Development & Implementation of EESOA

- It is fixed that the server, which contains the EESOA, will be first server in the list of host ips whose databases are to be used to fetch question for a particular test.

7.5.5 QSGLA Receive & Query Module

This module is responsible, which works in favour of the local agent - QSGLA, for receiving the parameters from the QSGMA and communicating with the particular DBMS in the particular distributed database server for the retrieval of the questions with the patterns specified by the parameters. The QSGLA is actually knows about the way to communicate with the DBMS in that particular server. The following figure (Fig. 7.9) represents the workflow of this module.
The working principle of this principle is described as below:

- The module receives the parameter list passed by the QSGMA, which contains the basically the question pattern and numbers of questions to be fetched of that particular kinds of pattern.

- Then question pattern related values are stored in the internal structure “QFP” whose structure is already discussed in “QSGMA Query Module”.

- Now, the module fetches the questions from the relation, named “QUESTION_MASTER”, according to the given criteria for the different levels of toughness/complexities. The structure of the relation “QUESTION_MASTER” is given below:
  
  - question_id
  - question_text
  - course_id
  - paper_id
  - unit_id
  - option_1
  - option_2
  - option_3
  - option_4
• The fetched questions are then stored in the internal structure “QF” (already discussed in case of the "QSGMA Query Module").

• The module then prepares a return list for the QSGMA which consists of the following items:
  
  o Agent Id
  
  o Question Id
  
  o Question Text
  
  o Option 1
  
  o Option 2
  
  o Option 3
  
  o Option 4
  
  o Correct Answer
  
  o Feedback
7.5.6 QSGMA Receive Module

This module works when it receives the return-list, from the local agent, with the item values mentioned in the "QSGLA Receive & Query Module". The work-flow of this module is depicted in Fig. 7.10. This is a very simple module and its working principle is given below:

- This module on receiving the return-list from the QSGLA, it populates the internal structure "QF" (described in "QSGMA Query Module") with the values in the return-list.
- It then updates the "DDE_TDH" internal structure.

7.5.7 QSGMA Migration Module

This module is responsible for the migration of the QSGMA to the next host described in the distributed database environment structure. Fig. 7.11 represents the work-flow of this module. The working principle of the module is:
This module consults the “DDE_TDH” internal structure and gets the information about the next host to migrate.

The QSGMA, communicate with the “JADE ACC/MTS Module-3” for migration.

The state of the agent is set as “stopped” on the local container of the distributed database server.

The agent, QSGMA, moves to remote container in the host indicated by the next host information.

The state of QSGMA is sets to “executing” and it starts executing on that container.

Figure 7.11: QSGMA Migration Module
7.6 Implementation of 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 [14,15].

```java
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 objects after fetching from the servers. Question is a simple class [18]:

```java
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 [18].

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 [18].

\[ \text{identifier} = \text{localname} + "@" + \text{platformmachine:port} + "\text{/JADE}" \]

The setup() method, is intended to include agent initializations, and is used to [18]:

- (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

```java
protected void doDelete() {
    //\}
}
```

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

```java
protected void takeDown() {
    //\}
}
```

The takeDown() method is used for any cleanup (e.g. unregister from DF) before the agent is destroyed [18].

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 [18].

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

    public boolean doneQ {
        
    }
}
```

The action() method [16] 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.

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 [18]:

```java
QFetchBehaviour hBehaviour;
```

Inside the setup() method, the statement,

```java
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 "|" [18].
Chapter 7: System Development & Implementation of EESOA

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 [18].

7.7 Implementation of 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 [18]:

```java
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 [18].

```java
protected void setup() {
    ..............................................
    ..............................................
}
```
The setup() method is intended to include agent initializations. In the setup() method, the method doListen() is called which will listen for the requests from agent QSGLA [18].

```java
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 [18].

```java
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 [18].

```java
public class QRetBehaviour extends SimpleBehaviour {
    //Code of the Constructors
    .........................................
    .........................................
    //Code of the different methods
    public void action() {
        ACLMessage msg = myAgent.receive();
        if (msg!=null){
            .........................................
```
Chapter 7: System Development & Implementation of EESOA

return true;
}
else {
    block();
    return false;
}

public boolean done() {

}

An agent can pick up messages from its message queue by means of the receive() method [17]. The block() method marks the behaviour of the agent as "blocked". So, inside the QSGLA agent class, there is a member-variable[18]:

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 [18].

REFERENCES:


