Chapter 6: Architecture & System Design of EESOA

This chapter explains the architectural framework of the web based Extended Expert System for Online Assessment (EESOA) of student learning. The term "extended" is used because the ESOA is extended using agent technology and distributed database technology (here, distributed database means not controlled by any centralized database management system).

6.1 Introduction

EESOA (Extended Expert System for Online Assessment), as already mentioned, is the extension of ESOA (Expert System for Online Assessment). The ESOA acts as the role of a teacher while assessing a student's knowledge about a particular topic/subject. Here the questions will be categorized into different level of toughness; the academic history of students will be categorized depending on some defined properties/characteristics. Also there will be rules for generating the question sets for students depending on their categories and for processing assessment results. Also there was a unique technical contribution, which is dynamic knowledge management, i.e. administrative users will be able to change the facts and rules of the ESOA with the help of interactive web-interfaces. [1]

In the ESOA, the student facts, question facts and the rules reside in the same server/platform where the ESOA runs. In India, there are different
universities and institutions. Most of them have their own database servers, containing a question bank, for their internal assessments/examinations. If a collaborative effort can be initiated among those universities/institutions so as to share their question databases then as a whole the question bank will be a very huge one.

In my proposed system, I have used the term “Distributed Environment” because here I want to mean the servers with different Database Management Systems but with same structure of the question databases. Also here we do not have any central Distributed Database Management System. Here the system will get question set from the various databases using agent technology.

Like ESOA, I also have used JESS for my research work. The prototype of this system is implemented and experimented in a J2EE environment using MVC paradigm. The system uses an external relational database for storing question databank and academic/performance history databank of the students. Here, the rule base of the system is stored in XML files.

As my system is the extension of ESOA, so basic Expert System part of EESOA is a rule based system.

### 6.1.1 Rule Based System

An expert system shell is nothing but just the inference engine. Most modern rule engines are specialized expert system shells having the features to support operation or programming in specific environment/domain.
A typical rule engine contains: an inference engine, a rule base and a working memory. The inference engine, in turn, consists of: a pattern matcher, an agenda and an execution engine.

In case of rule-based systems, rule and data are greatly associated with one another. A rule engine is for applying rules to data. So, the inference engine is the core part of a rule engine. It is the inference engine who controls the process of applying the rules to data contained in the working memory for obtaining the desired output from the system.

The pattern matcher component decides which rule/rules is/are to be applied when the current content of the working memory is given. Often the pattern-matching technique used by a particular rule engine will affect the kinds of rules written for that engine, either by limiting the possibilities or by encouraging us to write rules that would be particularly efficient.

When the inference engine finds out which rule/rules to be fired are stored on the agenda. The agenda is responsible for using the conflict strategy to decide which of the rules, out of all those that apply, have the highest priority and should be fired first.

Finally, when the rule engine decides what rule to fire, it has to execute the action part of that particular rule. The execution engine component fires the rules. In modern rule engines, firing a rule can have a wide range of effects. Some modern rule engines (like JESS) offer a complete programming language we can use to define what happens when a given rule fires. The
execution engine then represents the environment in which this programming language executes.

The rule base contains all the rules of the system. The rules can be stored as strings of text, but most often a rule compiler processes them into some form that the inference engine can work with more efficiently.

Some rule engines allow (or require) to store the rule base in an external relational database/text files or XML files and others have an integrated rule base. Storing rules in a relational database allows us to select rules to be included in a system based on criteria like date, time, and user access rights. JESS is a rule engine that supports this feature.

The working memory, also called the fact base, actually contains all the information that the rule-based system is working with. The rule engine maintains one or more indexes, similar to those used in relational databases, to make searching the working memory a very fast operation.

6.1.2 MVC Paradigm

MVC is an architectural pattern used in software engineering. The pattern isolates "domain logic" (the application logic for the user) from input and presentation (GUI), permitting independent development, maintenance and testing of each. The figure 6.1 shows how the three different logical functional blocks work together [2]:

The Model: The $M$ in MVC refers to the data object model [2].

The View: The view is responsible for presentation issues. It handles how the client will see the application [2].

The Control: The control part of the paradigm deals with the business logic of the application. It handles how and when a client interacting with the view is able to access the model [2].

6.1.3 J2EE Framework

The J2EE platform specifies the logical application components within a system and defines the roles played in the development process. J2EE, introduced in 1998, defines a multi-tier architecture for enterprise information systems (EIS). By defining the way in which multi-tier applications should be developed, J2EE reduces the costs, in both time and money, of developing large-scale enterprise systems [2].
6.1.3.1 Application components

Four application components are defined within the J2EE platform. They are as follows [2]:

✓ Application clients (Standalone Java clients): Clients are generally stand-alone applications written in Java. They run within a virtual machine and can use the J2EE standard services to access components located within another tier.

✓ Applets (Java code which executes within a browser): Applets are similar to application clients, but execute within a Web browser. Initially applets garnered extensive attention, as they were seen as a means of making Web pages more dynamic.

✓ Web components (JSPs, Servlets): Web components are server-side components, generally used to provide the presentation layer to be returned to a client. Two types of Web components exist: Java Server Pages (JSPs) and Java servlets.

✓ Server components (EJBs, J2EE API implementations): Server components come in the form of Enterprise JavaBeans (EJBs). EJBs execute within a container that manages the runtime behavior of the EJB. EJBs are usually where the business logic for an enterprise system resides.

J2EE's architecture maps onto the MVC nicely. Typically, entity beans are used to provide the model logic, while a mix of entity beans and session
beans are used to provide the control logic, and Web components are used to implement both control and presentation logic. In practice, however, the separation of the three types of logic is not as distinct, and additional patterns are often needed to support the development cycle. [2]

6.1.4 Java Expert System Shell (JESS)

JESS stands for Java Expert System Shell. It is a rule engine and scripting environment written by Ernest Friedman-Hill in Java language at Sandia National Laboratories, Livermore, Canâda. A program written in JESS may consist of rules, facts and objects. The inference engine decides which rules should be executed and when. A rule based expert system written in JESS is a data-driven program where the facts, and objects if desired, are the data that stimulate execution via the inference engine. [3]

JESS provides support for the modular development and execution of knowledge bases with the defmodule construct. JESS modules allow a set of constructs to be grouped together such that explicit control can be maintained over restricting the access of the constructs by other modules. [3]

Besides helping us to manage large numbers of rules, modules also provide a control mechanism: the rules in a module will fire only when that module has the focus, and only one module can be in focus at a time. [3]
6.1.4.1 JESS Facts and Rules

For a rule-based expert system, facts and rules are mandatory to make decisions/inferences/results. In JESS, facts can be stored in the working memory in three forms: (a) ordered facts (b) unordered facts and (c) shadow facts. Our proposed system will use shadow facts [4]. Shadow facts are just unordered facts that serve as “bridges” to Java objects [4]. Shadow facts are useful as because using this Java objects can be put into the working memory of JESS.

The system, EESOA, works with two kinds of facts:

- the student facts, which mean the students’ academic/performance history and

- question facts, which mean the questions.

MySql/PostgreSQL databases are used to store these facts and its integration with JESS is done by using JSP and Servlet technology. Also, by using JavaBeans, I have implemented the shadow fact technique.

A Jess rule is something like an “if... then” statement in a procedural language, but it is not used in a procedural way. While “if... then” statements are executed at a specific time and in a specific order, according to how the programmer writes them, Jess rules are executed whenever their if parts (their left-hand-sides or LHSs) are satisfied, given only that the rule engine is running. This makes Jess rules less deterministic than a typical procedural program. [4]
In our system, the rule data will be stored externally in XML files. Rules can be generated dynamically from XML files and loaded into the rule base of JESS.

6.2 Agent Based Systems

Earlier, the introduction of powerful abstractions, e.g. procedural abstraction, abstract data types and most importantly the object oriented programming, for managing the inherent software complexity resulted significant improvements in the field of software engineering. Now-a-day, scientist and researchers are trying to view the computer systems in terms of autonomous agents. And so the agent model is now considered as a new theoretical as well as practical model of computation. This is the reason, why it is being advocated that agents are the next generation model for engineering complex and distributed systems [5,6].

Without discussing the concept of MAS, the discussion of the agent based system will be incomplete. As mentioned in chapter 3, the need of agent-to-agent communication gives the birth of the concept of MAS (Multi-Agent System). Basically, multi-agent systems are developed to solve large and complex problems which includes the problems like transportation, power supplies etc. Also, in case of shopping through internet, i.e., e-commerce, cooperative agent-based systems are developed. To support personalised works, e.g., scheduling and shopping through internet (e-commerce), cooperative agent-based systems are developed.
In EESOA, two kinds of agents, namely mobile agent and static agent are used. The question fetching task from different question database servers plays a very important role in case of online assessment expert system and in EESOA, this task is carried out using agents.

6.3 Design & Methodology of EESOA

6.3.1 JADE with 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 (as discussed chapter 5).

6.3.1.1 JADE 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) [8].
6.3.2 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. [9]

6.3.3 Mobile Agent Design Pattern Used in EESOA

In the design and implementation of the EESOA [9] 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.

I 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, figure 6.2.

6.4 Architecture of EESOA

As we know, our previous architecture, i.e., ESOA, includes mainly five components [1]:

(i) Apache Web Server: listens for web page request

(ii) Tomcat Servlet Engine: serves dynamically generated web page using JSP and Servlet technology
(iii) MySql Database: stores the student information and the question bank of various courses.

(iv) XML: files for external Rule Base

(v) JESS Engine: for online assessment

In EESOA, the components (i), (ii), (iv), (v) of ESOA remain the same. The student information database part of the component (iii) resides in the same server where the EESOA resides but in case of the question bank, it is
a collection of question banks from different servers of different universities/institutions. So, there are two additional components in EESOA (figure 6.3):

(vi) JADE: for agent platform

(vii) JIPMS: for agents' mobility support

(viii) PostgreSQL: also used for storing the question bank of various courses.

Figure 6.3: Architecture of Extended ESOA
6.5 Working Principle of EESOA

- When a student appears in a self assessment or a conducted assessment, he/she will log-in through web-browser.
- He/she then should request EESOA to start the pre-assessment processes with required/related inputs through web-browser.
- EESOA control unit will handle the request and then instruct the EESOA model unit to retrieve student's fact data and rules from XML files and then passes to EESOA Engine.
- The EESOA engine then tries to evaluate the student's assessment question set pattern depending on the student's academic and performance history.
- Now the mobile agent will be started with the evaluated question set pattern for the assessment as parameter as well as with other different related parameters and its itinerary.
- The mobile agent will then move the first agency/machine and contact the local agent and passes its requirements.
- After getting the requirements passed by the mobile agent, the local agent contacted the local DBMS and fetches the questions.
- The local agent then gives this question set to mobile agent and the mobile agent then populates its question bank with this set and
suspends its own behaviour execution and moves to next agency/machine in its itinerary.

- Thus after travelling through each and every agency/machine according to the itinerary, the mobile agent comes back to the originating agency/machine (the EESOA server) with the fetched question bank and populates a table (figure 6.4) in the server as well as indicates its return to the EESOA engine.

- The question set retrieved by the mobile agent, QSGMA, which is inserted in a MySQL table (after coming back to home agency/machine). A screenshot of which shows a portion of the table populated by the mobile agent on arriving it's home:

```
<table>
<thead>
<tr>
<th>questionid</th>
<th>agentid</th>
<th>hostip</th>
<th>questiontext</th>
<th>option1</th>
<th>option2</th>
<th>option3</th>
<th>option4</th>
<th>answer</th>
<th>toughnesslevel</th>
<th>feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000009</td>
<td>mal@192.168.0.12:81/JADE</td>
<td>192.168.0.2</td>
<td>What is the RDBMS terminology for a column or field?</td>
<td>Tuple</td>
<td>Relation</td>
<td>Attribute</td>
<td>Domain</td>
<td>3</td>
<td>UE</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

Figure 6.4: Sample screen shot from “from_agent_temp” table

The hostip field (Figure 6.4), describes the host from which server machine a particular question is fetched by the mobile agent described by the agentid field value for that particular question.
The above table is further merged to the assessment table and used by the EESOA for conducting assessment respective assessment.

- EESOA Engine then loads the question facts, rules into its working memory (Figure 6.5), rule base and informs the administrator.

```jess
(facts)
f-0 (MAIN::initial-fact)
f-1 (MAIN::assessment (std_id 1) (course_id 3) (unit_id 1) (question_id "00000009") (question_text "What is the RDBMS terminology for a column or field?") (option_1 "Tuple") (option_2 "Relation") (option_3 "Attribute") (option_4 "Domain") (answer 3) (compl_level "VE") (agent_id "raal@192.168.0.12:81/JADE")
f-2 (MAIN::assessment (std_id 1) (course_id 3) (unit_id 1) (question_id "00000010") (question_text "What is the RDBMS terminology for a row?") (option_1 "Tuple") (option_2 "Relation") (option_3 "Attribute") (option_4 "Domain") (answer 1) (compl_level "VE") (agent_id "mal@192.168.0.12:81/JADE")
f-3 (MAIN::assessment (std_id 1) (course_id 3) (unit_id 1) (question_id "00000018") (question_text "What is the RDBMS terminology for a table?") (option_1 "Tuple") (option_2 "Relation") (option_3 "Attribute") (option_4 "Domain") (answer 2)
```

Figure 6.5: Sample screen shot from working memory of JESS (question fact)

- The EESOA Engine then instruct the EESOA model unit to store this question set with the student identification information into MySQL database and frees memory, held by the fetched question set, from the working memory.

- The EESOA Engine then call EESOA view/model unit to view the question set to the student to answer and keep track of his/her answers.

- When the student completed his/her examination, EESOA control/model unit will load the question set given to the student and his/her respective answers from MySQL database to working memory of the EESOA Engine and tells the EESOA Engine to processes the assessment result.
The EESOA Engine, after processing the assessment result, generates some feedbacks and EESOA model unit to store the student’s performance/given feedbacks into the MySQL database and send the result/feedbacks to EESOA view unit for the student to view.

The EESOA Engine then frees the memory held by the student academic/performance history from its working memory.

REFERENCES:


