This chapter discusses the prototype design, implementation and testing of ESOA. The prototype expert system is tested and evaluated with some sample data with the student model and a question bank of few subjects.
There are two major users in the system: the student on the client side and the teacher on the server side. So, there are two sets of use cases in ESOA: client side use cases and server side use cases.

### 8.1 Client-side use cases

There are three types of student’s interactions with ESOA. In Fig. 8.1, the client side use case diagram is shown.

![Client side use cases diagram](image)

*Figure 8.1: Client side use cases*

Here, user (student) interacts with the system with ‘authentication’ i.e. logging in for a new session. After authenticated by the system, the assessment process starts. In ESOA, each subject is divided into some chapters/units. In assessment, the student has to choose the chapter(s)/unit(s) for assessment. Then the system presents the questions to the students and the students answers the questions and submits them. This event triggers ‘validation’, the use case in which ESOA checks and updates the student model by estimating the student knowledge.
8.2 Server-side use cases

Teachers are in the server side. The following figure demonstrates the server side use cases:

![Server side use cases diagram]

Fig 8.2: Server side use cases

8.3 The ESOA Framework and components

The components of the ESOA are:

(a) The test edition module: It is used by domain experts to define the tests and the structure of the subject domain: topics, questions, relationships between them, and relative weights of topics in the test. This information is stored in the question knowledge base. In this module the domain expert can also define test specifications that will guide the item selection process, and the finalization criterion (maximum number of questions to be posed, minimum number of questions of each topic etc.).
A student model, which is created and updated by ESOA for each student that takes the test. In addition, the student model stores information about which questions have been asked by the system.

The expert assessment is the main module of ESOA. It is responsible for selecting the questions that is posed to each student. The generation process is guided by the temporary student model and the test specifications.

ESOA can be used in two different ways:

(a) The teachers and/or the domain expert can use it to develop the online tests i.e. to define their topics, questions, parameters and specifications.

(b) The questions in the assessment processes are presented to each student based on their knowledge level by the expert system according to the specifications provided by the domain expert/test designer. As the students answers the questions, the new ability level is computed by the expert system and the next question selected, until the stopping criterion is met.

Let us discuss the question knowledge base and the student model of ESOA.

8.4 The question knowledge base.

A database of questions of various units of various subjects is designed in MySQL. We have considered only multiple choice type questions for assessing the student’s knowledge. Each question in the question item
bank is characterised by a difficulty parameter having the values mentioned as:

\[ F = \{\text{very easy'}, \text{easy'}, \text{medium'}, \text{hard'}, \text{very hard'}\} \]

These difficulty parameters are determined and assigned by the subject experts of the domain.

8.5 Student model

In our system, student model is used for representing student knowledge and behaviour. In ESOA, we have used the overlay student model (discussed in chapter 2).

8.5.1 Student knowledge model

Information about student knowledge is gathered during tests. We considered the tests taken by individual students to estimate the student knowledge state in chronological order. Besides a detailed statistics about the student work, our system offers two reports in graphical form: overall progress report and objective wise progress report.

Objective-wise progress report shows the level of the student knowledge regarding each topic. A problem may arise when constructing this report, which is due to the possible repetition of the same question through the taken tests. Furthermore, answers given for the same question may vary from test to test. The actual state of student knowledge related to the topic is reflected by considering only the lattermost answer. However, in this case students would repeat tests until obtaining the maximum score of each topic. Consequently,
we opted for considering all the repetitions of questions in the construction of the objective-wise progress report.

8.5.2 Student behaviour model

We present the patterns considered:

- the location where they used the system from (inside/outside university)
- total time for self-assessment
- number of solved tests
- number of answered questions
- students’ preferences for different types of test

8.6 User interface

The system may be used by two types of users through the user interface: students and domain expert/teachers. This section deals with the operations for these users.

8.6.1 Student user

The system can be used in chapter following mode or free navigation mode. The chapter following mode is very simple, in this case students can view their performance and go further. On the other hand, there are five possible operations in free navigation mode. A student can choose from the following operations:

- Take test – this operation offers the student the possibility to select the desired test type.
- Continue test – as practice tests may be interrupted whenever students want, this option offers the continuation of tests started
previously. Student performance will be updated only when the test is finished.

- View test – students have the possibility to review their own tests already finished.
- Progress report – students may select to check their overall or objective-wise progress report.
- Statistics – this option shows some statistical data about the student work in the system.

### 8.6.2 Domain Expert/Teacher user

Teacher users may choose from the following operations:

- Question administration – this is used to manage the item bank.
- Test administration – used for creating, activating or deactivating tests.
- Most missed questions – this operation contains a table with the most troublesome questions for students. It is both a feedback from students and a very useful operation to calibrate the difficulty level of the questions.
- User Profiles – the teacher can see all the information about a selected user. It is used to show the teacher the complete profile of a student. It is useful when analysis is intended regarding the behaviour of a particular student.
- Statistics – a lot of useful statistical data is offered about student results and tests.
8.7 System Design and Prototype Development

8.7.1 Software Process Model Used

We have used the incremental software process model to develop the expert system.

8.7.2 Software Tools Used

We have implemented ESOA in a J2EE framework using MVC paradigm. The following software tools are used in the design of ESOA.

- JDK 1.5
- Apache Web Server 2.2
- Apache Tomcat Server 6.0
- MySQL Server 5.5
- Mysql-connector-java-3.1.12
- JESS 7.1P2

Since for the prototype design of ESOA, we have used JESS as the rule engine and since Jess is a programmer’s library written in Java; therefore, to use Jess, we'll need a Java Virtual Machine (JVM). JESS includes a JESS.jar file. To use JESS as a library from our Java programs, the file jess.jar (in the lib directory) must be added in the CLASSPATH variable list. We have configured all the necessary steps to work JESS properly.

8.7.3 Assessment Strategy

In ESOA, after logging to the system, the student has to select the course first. Then depending upon the course selected by the student, the subjects of that course are displayed to the student again. The
student has to select the subject again and after selecting the particular subject, the chapters/topic of that subject will be displayed to the student for taking the self-assessment.

In the implementation of ESOA, we have considered a large number of rules where the rules are fired depending upon the student model. i.e., each time the inference engine has to search for the student model for the assessment history of the students and accordingly the questions are presented to the student for each attempt. In section 5.4, we already mentioned that in ESOA, no factual knowledge are used as they are used in most of the traditional expert system development. Here, we have assumed that the rules are designed to emulate the techniques/rules applied by an expert/teacher in a classroom. So, in the implementation of the knowledge base of ESOA, rules are designed on the basis of heuristic knowledge. But, the knowledge base can be modified/extended according to the advice/strategies of the domain expert.

The rules are categorized in to three types: student-history-assessment rule, question-generation rule and answer-check rule. There are several rules for each of the type/category.

Below is the pseudo code of a rule from the question-generation rules:

Question-Generation-Rule 1:

If assessment-attempt = 1, then

generate 40 questions of the 1st chapter with toughness-level(VE = 30%, E = 20%, M = 20%, T = 20%, VT = 10%)

When a student completes his/her assessment attempt, then an answer-check-rule from the rules will be applied for the further
assessment. For example, if a student does not able to answer 80% of the questions correctly, then he/she cannot proceed to the next chapter for assessment. In the different answer-check-rules various factors like: Correct answers (in percentage) for each of the toughness-level, the percentage-gap between the percentages of the correct answer for the nearest two toughness levels etc. are considered.

### 8.7.4 Implementation of JESS Rules with Java

Since for the prototype design of ESOA, we have used JESS as the rule engine and since Jess is a programmer's library written in Java; therefore, to use Jess, we'll need a Java Virtual Machine (JVM). When JESS rules are executed in Java, JESS library files are also to be loaded into the class path of Java. In JESS two JAR files should be included in order to execute a Java file which has JESS commands embedded in it.

Instance of JESS Rule engine can be created in Java code. Instance of JESS can then be reused to process each assessment task. JESS Rule Engine is having special API to execute rules. For Example:

```java
Rete engine = new Rete();
Engine.batch("rule1.clp");
Engine.run();
```

### 8.7.5 Representation of Facts

We have grouped together rules and facts that together operate on the same stage of the program’s execution. It allows us to give “focus” to specific modules, thus temporarily enabling and disabling groups of rules and facts.
We have used two types of facts in the expert system:

(a) student model (student facts):

(b) questions database (question facts).

Each question is tagged with one from the 5 types of toughness; very easy (VE), easy (E), moderate (M), tough (T) and very tough (VT). The assignment of the toughness of each question is determined by the domain expert/teacher.

Since we are using JESS as the rule engine, let us take a brief overview of how facts are represented in JESS's working memory. We can design some pure facts defined and created entirely by Jess. Other facts are shadow facts connected to Java objects we provide. Shadow facts act as "bridges" that let Jess reason about things that happen outside of working memory.

To accumulate the facts, first of all we have to create a template. A template consists of two parts: a name and a set of slots. The structure of a template is like the class of a Java object, or a table of a relational database. The slots can represent the properties of the JavaBean, or the columns/fields of a table. A fact is therefore like a single JavaBean, or like a row in a database table.

The facts can be represented by 3 ways: (a) Ordered facts (b) Unordered facts (c) Shadow facts

The following example shows that structure of some template used in the design of the fact base which can be represented by the ordered facts:
(deftemplate course (slot course_id) (slot course_name) (slot duration) (slot stream))

(deftemplate paper (slot paper_id) (slot paper_name) (slot course_id) (slot semester) (slot no_of_unit))

(deftemplate question (slot course_id) (slot paper_id) (slot unit_id) (slot q_no) (slot q_type) (slot q_text) (slot op1) (slot op2) (slot op3) (slot op4) (slot c_op) (slot compl_level))

But if we use the above method to store the facts, the size of the working memory will be very large. Since in ESOA, we have to design the student model and the question data bank; therefore, all these facts are maintained externally in a MySQL database. Since JESS’s working memory is used to store the facts of the expert system, we can store all the facts externally either in the form of some text (.clp) file or in some xml file or in some RDBMS. For the better management of the data, we have stored the data in a MySQL database. Using JSP and Servlet technology and JavaBean, we have integrated the MySQL database with JESS and loaded the data in the form of facts into the working memory of JESS. Here JavaBeans are used to implement the shadow facts.

Following is the structure of the student model (assessment history) used in ESOA. In each assessment process, this model is accessed by ESOA and updated.
Table 8.1: Student model (TEST_MASTER table)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_id</td>
<td>string</td>
<td>Unique Test Id</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>semester_id</td>
<td>string</td>
<td>Unique Semester 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>chapter_id</td>
<td>string</td>
<td>Unique Chapter Id from CHAPTER_MASTER</td>
</tr>
<tr>
<td>test_type</td>
<td>char(2)</td>
<td>Test Type</td>
</tr>
<tr>
<td>date_of_test</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>
<tr>
<td>test_end_time</td>
<td>Date/Time</td>
<td>Test End Time</td>
</tr>
<tr>
<td>total_questions_given</td>
<td>integer</td>
<td>Total No. of Questions Given</td>
</tr>
<tr>
<td>total_correct_answers</td>
<td>integer</td>
<td>Total No. of Questions Correctly Answered</td>
</tr>
<tr>
<td>result</td>
<td>string</td>
<td>Result of the Test</td>
</tr>
<tr>
<td>remark</td>
<td>string</td>
<td>Remark of the Test</td>
</tr>
</tbody>
</table>

Table 8.2: Student model (TEST_QUESTION_ANSWER table)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_id</td>
<td>string</td>
<td>Unique Test Id from TEST_MASTER</td>
</tr>
<tr>
<td>std_id</td>
<td>string</td>
<td>Unique Student Id from STUDENT_MASTER</td>
</tr>
<tr>
<td>question_id</td>
<td>string</td>
<td>Unique Question Id from QUESTION_MASTER</td>
</tr>
<tr>
<td>op_correct</td>
<td>char(1)</td>
<td>Correct Option Selected</td>
</tr>
<tr>
<td>correct_or_not</td>
<td>char(1)</td>
<td>Answered Correctly or Not</td>
</tr>
<tr>
<td>question_toughness</td>
<td>char(2)</td>
<td>Toughness Level of the Question</td>
</tr>
</tbody>
</table>
8.7.5.1 Shadow Fact Representation

The shadow facts are unordered facts and are similar to JavaBeans. Shadow facts serve as a connection between the working memory and Java application in which JESS is running. The function `defclass` is used to create a `deftemplate` which is used to create a `deftemplate` in JESS. The `definstance` commands 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 similarity between javabean and unordered facts as both have a list of slots which can hold values and these values can change at run time. For a javabean, these slots are called properties. A javabean property is normally a pair of methods to `get` and `set` the property values. 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 Fig 8.3 and creates a `deftemplate`.

We have used the `Introspector` class to automatically generate a `deftemplate` that represents any specific JavaBean class.

![Figure 8.3: Shadow fact implementation using JavaBean](image-url)
Creating a deftemplate in the rule engine

The syntax is as follows:

(defclass Student project.st.Student)

This defclass when loaded creates a deftemplate by name Student with slots corresponding to the student bean properties

8.7.5.2 Representation of Rules

In ESOA, the rule base is designed using JessML, an XML-based rule language. JessML is a hybrid imperative/declarative language, just like the Jess language itself. 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 lhs 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:

(defrule StdAcHPerH1

  ?acdh <- (student (std-id ?sid))


  =>

  (calculate_performance ?acdh ?pph ?type)
)

For example, following is an example of student-history-assessment rule:
The above rule will be triggered when a student wants to attempt any of the four tests: standard test, practice test, objective-wise test and custom test. When a student logs-on to the ESOA successfully, then he/she can go for a test according to his/her choice. For taking a test, the student must have to select:

(i) the test type, (ii) the course/subject and (iii) the chapter.

After all the necessary selections have been done, he/she can go for the test. At this moment the above rule, StdAcHPerHl rule, is triggered. Here in this rule, the student’s academic history and his/her performances of previous attempts for the same type of test are retrieved from the JESS working memory. These retrieved information along with the test-type are sent as parameters to the JESS function, calculate_performance, which is called when the retrieval completes.

Following is an example of question-generation rule used in the system:

(defrule QBfetchRule1

   (declare (salience 1))

   ?student <- (student (and (achlemp ?achlemp&:(and (> = ?achlemp 30) (< ?achlemp 40))) (achstr “Science”)))

   =>

   (extractqb 30 20 20 15 15))

In the above rule: QBfetchRule1 is the name of the rule, achlemp refers to the slot name which indicates the percentage of marks obtained by the student in the last examination, achstr refers to the slot which indicates the stream of the student’s in the his last degree examination, extractqb is a deffunction which will be called depending upon the
conditions in the LHS of the rule and it has five parameters for the toughness of the questions to be fetched according to the toughness – Very Easy(VE), Easy(E), Medium(M), Tough(T) and Very Tough(VT) respectively.

8.8 Querying Working Memory

Jess's working memory is similar to a database; it's filled with indexed, structured data. Most of the time, we access working memory by pattern matching from a rule.

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

A time-honored way to search a collection of items is by using linear search and a filter, a Boolean function that indicates whether an item should be part of the search result or not. Jess supports this kind of brute-force search of Java objects in working memory using the jess.Filter interface. We can implement jess.Filter and then pass an instance of the filter to the jess.Rete.getObjects(jess.Filter) method, which will return a java.util.Iterator over the selected objects. Since this is not practical for a large working memory sizes, we have not used this method.

We have used the defquery construct for query the working memory.

8.9 Evaluation of Expert System

The evaluation process is common to all areas of science and technology. Evaluation is a process for continuous revision of a product during the cycles of development. The point is how does the evaluation of expert systems differ from general evaluation processes? The
evaluation step in the cycle of expert systems development includes the following activities: (a) Field Test (b) Refinement and (c) Implementation.

(a) **Field Test:** Field testing activities in expert systems development are complicated by the fact that there may be no formal way to prove an answer is the correct or best solution. User acceptance is a critical component of evaluation. If an expert system is unfriendly, confusing and tedious; then it will be unacceptable to users regardless of performance results (Waterman, 1986). Factors which might be considered in field testing include utility, flexibility, ease of use, intelligibility of output, speed, efficiency and reliability. A review of cases or problems by experts assures the knowledge engineers that the expert system is responding to an appropriate range of problems in similar fashion to expert’s response pattern. Blind reviews in which the best domain experts compare solution obtained by the machine and human generated solutions without knowledge of machine involvement are highly recommended for final field tests (Parry, 1986)

**Field Testing of ESOA:**

The developed prototype of ESOA is field tested to determine needed changes. Since field testing is simply executing cases or problems which are selected during the design of the expert system, both typical and unusual cases are tested to identify the strengths and weakness of ESOA. A dummy question database and a student model is created and the system is tested.

**Refinement:** It refers to the continuous refinement of the expert system. Knowledge engineers depend upon evaluation for testing and improving their bases and the functioning of the expert system. Domain
experts must access how their knowledge is being used in the system. Evaluation promotes communication between the knowledge engineer and the domain expert. More knowledge is added to the expert system through the continuing process of redesign and prototype revision. Several different prototypes should be created during the usual course of expert system development. Then final revisions may be made prior to implementation in the real world setting. In case of ESOA, with the incremental software process model, the prototype system is redesigned and tested with different fact data and rule data. In our research work, ESOA is tested with few rules but there are provisions of adding some new rules to the system to make it more knowledgeable.

**Test Cases and Test Implementation:**

For the testing of ESOA, I have considered the Master of Computer Application (MCA) course as a test case. The course is comprised of a large numbers of papers. I have considered the paper named "Programming in C". In the design of the question bank, the paper is divided into 8 units. Depending upon the toughness level of the questions, each unit is organised with some multiple choice objective type of questions. A dummy student model is created. With the student model, the system is tested with different user inputs and according to the rule base of the system; each dummy student receives different sets of questions according to the previous attempted history.
A sample user interface for assessing student knowledge in the system is shown in Figure 8.4.

Figure 8.4: User Interface for assessment