Chapter 5: Implementation of MASMEE

5.1. Tools and Techniques used in Evaluation

MASMEE is implemented in Java Programming language as a prototype application in six incremental phases namely, Test Paper Generation, Subjective Evaluation Module, Objective Evaluation Module, Practical Evaluation Module, MASMEE Agent model and Web Interface. Several tools, libraries and techniques/technology used in different phases are shown in Table 5.1 (refer to Figure 4.2, to recall the methodology of MASMEE). The programming of all modules is performed using Java Programming Language. Tools and libraries have been used to perform several tasks. These tools are invoked from main Java program in Examiner Agent module. In Hybrid technique, the pre-processing steps of tokenization and stop word removal are implemented using regular expressions. The synonym search is performed using WordNet tool and JWI library. Latent Semantic Analysis technique is implemented using Guava Library and MatLab. Guava library is used to calculate frequency of words in student answer. The Singular Value Decomposition (a step in LSA) is performed using MatLab. Fuzzy Logic is implemented in MatLab. MatLab Control Library is used to invoke MatLab from Java Code. Subject Specific Ontology is implemented using Protégé tool. Apache Jena Library and Sparql 1.1 are used to extract information from Ontology. External library is used for Maximum Entropy. Objective evaluation module uses WordNet tool for synonym search in fill-in-the-blanks and BLEU technique for evaluation of one-line answers. In Practical Evaluation module, compilation is done using Turbo C++, DosBox and JDK. Testing is performed with the help of CUnit, CppUnit (MingW) and JUnit. Metrics are calculated using CCCC tool. The Program Dependence Graphs are generated using Frama-C and SDG Library. The similarity between PDG is found by implementing Rapid Subgraph calculation algorithm in Java Programming Language. MASMEE Agent model is implemented using Java Programming Language and Java Agent Development Environment (JADE). Swings and JFormDesigner are used for Examiner Agent interface design. Web Interface is implemented in Java Server Pages, HTML 5 and CSS 3. Apache Tomcat Server is used for deploying MASMEE system. The database is implemented in Oracle 12c. The Examiner Agent and Student Agent can access the database using SQL queries. The ODBC Drivers are used for connecting to the database. The development platform is Windows 8 (64-bit). The hardware
configuration of development platform is AMD Quad core 2GHz processor with 8GB RAM. The details of development of each phase are discussed in the subsequent sections.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Phase</th>
<th>Tool</th>
<th>Library</th>
<th>Technology/ Technique</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 0</td>
<td>MASMEE Programming</td>
<td>Java Development Kit 1.7 (JDK)</td>
<td></td>
<td></td>
<td>It is used for development of MASMEE</td>
</tr>
<tr>
<td>Phase 1</td>
<td>Test Paper Generation</td>
<td>Oracle, JDK</td>
<td></td>
<td></td>
<td>Implemented through coding using Java and Oracle</td>
</tr>
<tr>
<td>Phase 2</td>
<td>Subjective Evaluation</td>
<td>MatLab</td>
<td>MatLab Control Library [100]</td>
<td>Fuzzy Logic Matrix Operations</td>
<td>Used for LSA matrix calculations and Fuzzy Logic implementation for hybrid technique.</td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
<td>WordNet 2.1 [101], [102]</td>
<td>Semantic Networks</td>
<td>Used for finding word Synonyms</td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
<td>JW1 2.2.3 [103]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
<td>Guava library [104]</td>
<td>Multi Hash Lists</td>
<td>Used for counting frequency of words</td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
<td>Protégé 5.0 beta [105]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
<td>Apache Jena library [106], [107]</td>
<td>RDF format of ontology</td>
<td>Subject Specific Ontology Development</td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
<td>Maximum Entropy [34]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 3</td>
<td>Objective Evaluation</td>
<td>WordNet 2.1 [101], [102]</td>
<td>JW1 2.2.3 [103]</td>
<td>Semantic Networks</td>
<td>Used for finding word Synonyms</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td>Turbo C++</td>
<td></td>
<td></td>
<td>Compilation of C and C++ Program</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td>DosBox</td>
<td></td>
<td></td>
<td>To Run Turbo C++</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td>MingW</td>
<td></td>
<td></td>
<td>To run CppUnit for testcases of C++</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td>JDK1.7</td>
<td></td>
<td></td>
<td>Compile java program</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td>CCCC Compiler</td>
<td>SDG library [108]</td>
<td>Program Dependence Graph</td>
<td>To find metrics of Java, C and C++ programs</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td>Frama-C [99]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td></td>
<td>CUnit [97]</td>
<td>XUnit Technology</td>
<td>To execute test cases on student programs</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td></td>
<td>CppUnit [96]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 4</td>
<td>Practical Evaluation</td>
<td></td>
<td>JUnit [95]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 5</td>
<td>MASMEE Agent Model</td>
<td>Oracle</td>
<td>Java Agent Development Environment (JADE) [109]</td>
<td>For agent development and interactions Database is implemented in Oracle.</td>
<td></td>
</tr>
<tr>
<td>Phase 5</td>
<td>MASMEE Agent Model</td>
<td>Swings</td>
<td>JFormDesigner</td>
<td></td>
<td>Interface design</td>
</tr>
<tr>
<td>Phase 6</td>
<td>Web Interface</td>
<td>Java Agent Development Environment (JADE)</td>
<td></td>
<td>Apache Tomcat Server</td>
<td>Web Interface hosting</td>
</tr>
<tr>
<td>Phase 6</td>
<td>Web Interface</td>
<td></td>
<td></td>
<td>Java Server Pages</td>
<td>Server Side Scripting</td>
</tr>
<tr>
<td>Phase 6</td>
<td>Web Interface</td>
<td></td>
<td></td>
<td>HTML5</td>
<td>Structuring</td>
</tr>
<tr>
<td>Phase 6</td>
<td>Web Interface</td>
<td></td>
<td></td>
<td>CSS</td>
<td>Designing</td>
</tr>
</tbody>
</table>

Table 5.1 Tools and Techniques used
5.2. Implementation Details of Test paper generation

The test paper generation is implemented using Java Programming language. The main task was creation of database. For database, questions are collected for Computer Graphics (300) and Artificial Intelligence (75). The different parameters like difficulty level, time taken, marks, subtopic, etc are assigned to each question. This database acts as input to test paper generation program based on Randomized Selection algorithm discussed in section 4.2.

The code snippet showing implementation of Randomized Selection algorithm is shown in Figure 5.1. The input is subject, sections in test paper (equals number of subtopics), questions per section, difficulty level, type of paper (subjective, objective or practical) and type of objective question (one-line, fill-in-the-blanks and multiple-choice questions). First, the subtopic list is fetched from the subject table in database. The for each subtopic list of questions meeting the given difficulty constraints is fetched. The database is checked that it has as many questions as number of questions per section. As many random numbers are generated as the number of questions per section. These random numbers are sorted. Then select the question in question list at the index value of random numbers.

```java
public String[] returnTest(String subject, int sections, int quepersec, int diffic, String type, String specific)
{
    int TotalnumOfQue = (sections*quepersec);
    String subtopic[] = null;
    int numberOfQuestionperTopic = quepersec;
    int countSubtopic = 0;
    try{
        // get the list of subtopics
        String stmt= "select subtopics from subjects where name='"+subject+'"';
        ResultSet r1 = (ResultSet)myAgent.getDatabase(stmt,"Result");
        while(r1.next()) subtopic[r1.getString("subtopics").split(','));
        r1.close();
        countSubtopic = subtopic.length;
        int index=0;int indexTopic=0;
        // for each subtopic
        while(indexTopic<subtopic.length && index<TotalnumOfQue)
        {
            int j=0;
            String stmt1;
            // get list of questions that belong to given subtopic, difficulty, type and subject.
            if(specific.isEmpty())
                stmt1="Select question,marks from model where subject='"+subject+'" and subtopic='"'+subtopic[indexTopic]+'" and difficulty='+diffic+'" and type='+type+'"+type+'";"
            else
                stmt1="Select question,marks from model where subject='"'+subject+'" and subtopic='"'+subtopic[indexTopic]+'" and difficulty='+diffic+'" and type='+type+'"+type+'" and specific='+spec;
            ResultSet r11 = (ResultSet)myAgent.getDatabase(stmt1,"Result");
        }
    }
```
5.3. Implementation Details of Subjective Evaluation module

5.3.1. Tools used

The Subjective Evaluation module was primarily developed in Java Programming language. The implementation of Hybrid technique and Ontology based evaluation is performed by programming and using external tools and libraries. The tools used for subjective evaluation are briefly given below. More details of the tools are given in Appendix D:

- **WordNet**: It is generic lexicon Ontology which has a network of English language words connected by their relationship. It can be used to find the word context, synonyms, hyponyms, and calculation of semantic difference between two words.
- **MatLab**: Soft computing platform can be used for Matrix Operations and soft computing techniques like Fuzzy Logic.
- **Matlab Control Library**: It is used to integrate java programs with MatLab. It facilitates invoking and passing commands to MatLab engine from Java directly without user intervention.
- Guava libraries: They are open source libraries used for sorting and duplicate elimination in keywords. In addition, the frequency of each word in student answer is found using this package.
- JWI libraries: These are used to facilitate reading of WordNet data from Java program.
- Apache Jena library: It is used to read ontology, create its in-memory model and extract information after reasoning for required information using Sparql Queries.
- Protégé tool: The Subject Specific Ontology is developed using Protégé tool.
- Maximum Entropy Library: This library includes an implementation of Maximum Entropy technique for text classification.

5.3.2. Implementation of Proposed Hybrid Technique

The development of hybrid algorithm includes the following steps:

1. Tokenization and Stop word removal: The Tokenization and Stop Word Removal are performed by using the Spilt and Replace function of String class in Java Language using Regular Expressions. The stop word removal is shown in Figure 5.2. The various Stop words removed include a, an, as, the, of, it, is etc.

![Stop Word Removal using Regular Expressions](image)

2. Synonym Search: WordNet tool is used for synonym search. Java WordNet Interface (JWI) library is required to connect Java and WordNet. The details about WordNet tool
are in Appendix C. The Program Snippet for interfacing Java and WordNet through JWI library is shown in Figure 5.3. The Figure shows the JWI API used for accessing WordNet Data files. The entry point for accessing WordNet data using JWI library is the ‘IDictionary’ interface defined in JWI. First, it opens the WordNet dictionary using path of physical location of Dictionary (IDictionary). Second, the words have different context and belong to different parts of speech. Since the synonyms of words are required, all the variations of word – Parts of Speech like noun, verb, adverb, adjective, etc need to be extracted (POS). A loop is used to search in all contexts. Third, the word for which synonyms are to be found “search” is looked for in all POS tagging. The function ‘getIndexWord’ will find the specified word in WordNet Dictionary and return its index stored in ‘wordID’. Using the ‘wordID’ its synonym set (‘synset’) is identified. The most frequently encoded relation among synsets is the super-subordinate relation (also called ISA relation). Then for each sense of the word, the related and similar meaning words are fetched and returned as answer.

Figure 5.3 Program Snippet for WordNet and JWI usage

3. Stemming: It is performed using replace function in Java Programming Language using Porter’s algorithm.

4. LSA: The Latent Semantic Analysis algorithm is implemented in Java Programming language. The term frequency matrix, $A_{M,N}$ (in Figure 4.6) is calculated in Java
Programming language with the help of Guava Library. Singular value Decomposition (SVD) is performed in MatLab using MatLab Control Library. The cosine correlation is calculated in Java Programming Language.

a) Guava Library 18.0: The Guava project contains several of Google's core libraries that are used in their Java-based projects. It is open source library. The multi-hash library is used which helps in counting frequency of words in an array. The code snippet showing the ‘Multiset’ class is given in Figure 5.4. After tokenization, stop word removal, synonym search and stemming, the final list of keywords is ready. First, all the keywords are added to ‘ArrayList’ object. Then Multiset is created from this ‘ArrayList’. ‘Multiset’ is a hash map with count value for each key. The count value shows the number of times word is repeated in ‘ArrayList’. This is used to generate term frequency matrix for all answers.

```java
ArrayList<String> al = new ArrayList<String>(); // create a array list
for(String term : al) // term contains all tokenized, stop word removed, synonym searched and stemmed
    al.add(term); // add these terms to an array

Multiset<String> multiset = HashMultiset.create(al); // create a hash map from the array
al.clear(); // clear the array
for (Multiset.Entry<String> entry : multiset.entrySet())
{
    al.add(entry.getElement()); // add elements back from Multiset. The duplicates are removed.
}
```

Figure 5.4 Use of Guava Library

b) Singular value Decomposition using MatLab: After the term frequency matrix is made, Singular value decomposition is performed. There are inbuilt tools within MatLab that can be used to perform both of these decompositions. MatLab Control Library 3.01 is a Java API that allows for calling MatLab from Java. The usage of MatLab and MatLab Control library is shown in Figure 5.5. The MatLab Control Library packages to include are: ‘matlabcontrol.*’ and ‘matlabcontrol.extensions.*’. The ‘MatlabProxyFactory’, ‘MatlabProxy’ and ‘MatlabTypeConverter’ classes are
used. ‘MatlabProxyFactory’ are used to generate an instance of MatLab engine. ‘MatlabProxy’ invokes the engine. ‘MatlabTypeConverter’ helps in datatype conversions and row-major to column major array conversions. The array ‘A’ is passed to MatLab and SVD is performed. ‘SE’ and ‘d’ products are calculated in MatLab.

```
MatlabProxyFactory factory = new MatlabProxyFactory();
MatlabProxy proxy = factory.getProxy();
MatlabTypeConverter processor = new MatlabTypeConverter(proxy);

double[][] seq = processor.getNumericArray("SEQ").getRealArray2D();
rank = processor.getNumericArray("R").getRealArray2D();
for(int i=0; i<seq[0].length; i++)
    q[i] = new double[i][seq[0].length];
dh=0.0;
if(m==null) m=modelAnswer;
for(String value: m)
    for(int i=0; i<seq.length; i++)
        if(equlizer(value, masterTermVector[i]))
            for(int j=0; j<seq[0].length; j++)
                q[i][j] = q[i][j] + seq[i][j];

double[][] normQ = processor.getNumericArray("normQ").getRealArray2D();
for(int i=0; i<rank[0][0]; i++)
    for(int j=0; j<cola; j++)
        solution[i][j] = solution[i][j] + q[i][j] * d[i][j];
```

Figure 5.5 LSA related Code Snippet- MatLab Control Usage

c) Cosine Correlation code: The cosine correlation is calculated between two vectors ‘q’ and ‘d’ as shown in Figure 5.5. The SVD step of LSA generates term vectors and
document vectors. First, the term vectors for words belonging to model Answer are added (‘q’ in Figure 5.5). It generates the vector of keywords (‘q’). Then multiply this vector of keywords with document vectors (‘d’) to generate solution. This value of solution is divided by product of norms of ‘q’ and ‘d’ vectors.

5. BLEU: BiLingual Evaluation Understudy algorithm is implemented in Java Programming language. The frequency of each word is calculated using Guava Library and then the BLEU average is calculated. Code snippet implementing BLEU is shown in Figure 5.6. The variable masterFrequencyMmax contains frequency of each keyword in model answer. ‘maxMaster’ has total number of words in model answer. ‘bleuNumberOfWordsInAnswerWt’ variable has total number of words in each answer. ‘bleuRepOfKeywordInAnswerWt’ variable stores the frequency of each keyword in each student answer. Based on these four variables clipping of extra appearance is performed in BLEU.

```java
setMasterTermVector(model);
selAnswerTermFrequency();
double maxMaster = 0;
for(int z=0;z<masterFrequencyMmax.length;z++)
    maxMaster = maxMaster + masterFrequencyMmax[z];
value = new double[answersStudent.length];
double num=0, den=0;
for(int i=0; i<bleuNumberOfWordsInAnswerWt.length; i++)
{
    for(int j=0;j<masterFrequencyMmax.length; j++)
    {
        if(bleuRepOfKeywordInAnswerWt[j][i]>0.0)
        {
            if(bleuRepOfKeywordInAnswerWt[j][i]< masterFrequencyMmax[j])
                num = bleuRepOfKeywordInAnswerWt[j][i];
            else
                num = masterFrequencyMmax[j];
            if(bleuNumberOfWordsInAnswerWt[i]>maxMaster)
                den = bleuNumberOfWordsInAnswerWt[i];
            else
                den = maxMaster;
            System.out.printf("%s num=%-2f den=-%2f", masterTerms[j],num,den);
            value[i] = value[i]+ num/den;
        }
    }
}
```

Figure 5.6 Code Snippet implementing BLEU

6. Fuzzy Logic: The rules and membership functions for Fuzzy Logic based combination of output of LSA and BLEU are given in Design details of Subjective evaluation. Here its implementation is discussed. The fuzzy Logic toolbox available in
MatLab is used for its implementation. MatLab Control library is used for invoking MatLab from Java and passing the LSA and BLEU correlations to fuzzy toolbox. The fuzzy toolbox calculates the Final correlation value based on rules defined using rule viewer of fuzzy toolbox. The Figures showing implementation of Fuzzy Logic in MatLab are given in Figure 5.7 (a-d). The input variables are shown in Figure 5.7 (a) and (b), output variable in figure 5.7 (c) and fuzzy rules in Figure 5.7 (d). The Mamdani inference system is used for fuzzy inference.

Figure 5.7 (a) LSA Input Variable

Figure 5.7 (b) BLEU Input Variable
5.3.3. Implementation of Proposed Ontology Based Evaluation

The development and implementation of Ontology based evaluation involves: development of Ontology, extraction of ontology from RDF file and Implementation of Machine learning techniques in Java Programming Language.

The Ontology is implemented with the help of Protégé tool. The developed Ontology is included in Appendix C. The Sparql queries were written to access information from the Ontology. Jena API was used to access the Ontology from Java code. The machine learning techniques LSA and BLEU are used as implemented for Hybrid technique with some customization as discussed in this section. Generalized LSA is implemented in Java Programming language. Open source package is used for Maximum Entropy technique, available from SourceForge website.

1) Development of Ontology using Protégé: Protégé was used to develop the ontology for “computer Graphics” subject. Protégé is a free, open-source platform that provides tools to construct domain ontology. It supports a variety of formats for Ontology like XML, RDF and OWL. RDF format is used in this work, as it is fully developed and
supported by a number of applications. The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a format for Ontology and is similar to entity–relationship diagrams, as it is based upon the idea of making statements about resources (in particular web resources) in the form of subject–predicate–object expressions. These expressions are known as triples in RDF terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. Example, the Ontology subject – ‘Video_device_CRT_Monitor’ in computer graphics has predicate/property ‘hasResolution’. The value of ‘hasResolution’ is 640X480, becomes the object of this subject. Figure 5.8 (a-c) shows the ontology developed using Protégé. In Figure 5.8 (a) the Interface shown is Protégé tool. Details of Protégé are given in Appendix C. The Classes start from ‘thing’ which is main class of all the classes, beginning of everything. Under ‘thing’ all classes appear. The main class is Computer Graphics. Under computer Graphics all other classes are arranged. In the middle box, individuals belonging to each class are shown. In the third box, properties of individuals are shown. Figure 5.8 (a) shows the individuals designed for Computer Graphics Ontology. The ‘cross_sectional_view_design’ individual belongs to classes ‘computer_aided_design’ and ‘three_dimensional’. Figure 5.8 (b) shows the properties of individuals. The class ‘working_of_CRT’ has individual ‘Step_1_generation_of_electron_beam’. This individual has properties- ‘target’, ‘actor’ and ‘output_of_event’. The ‘target’ property symbolizes the need for the step in working of CRT. The ‘output_of_event’ is expected result of performing step 1. ‘Actor’ property is used to identify the elements involved in performing step 1. Figure 5.8 (c) shows part of ontology graph consisting of classes and subclasses.
Figure 5.8 (a) Individuals of Classes in Computer Graphics Ontology

Figure 5.8 (b) Properties of Classes in Computer Graphics Ontology
2) Extraction of Ontology from Ontology file: After the Ontology is developed using Protégé, it is stored in a file on the hard disk. Java programming language along with Apache Jena Library and Sparql queries are used to extract the ontology and use it for evaluation. Sparql 1.1 is a query language to access the knowledge saved in RDF triples of Ontology. It is similar to SQL. Apache Jena Library 2.13.0 is an open source Semantic Web Framework for Java Programming Language that helps to create an in-memory model of Ontology where Sparql can be executed on it. It provides API to extract from and write to RDF Graphs. The graphs are represented as an abstract model. A model can be sourced with data from files, databases, URLs or combination of these. A model can be queried through Sparql 1.1. Figure 5.9 shows Jena and Sparql used for
accessing information from designed and developed Computer Graphics ontology. In Figure 5.9, The Sparql query used to extract ontology detail is shown. As can be seen this query is like Select statement in SQL. Apache Jena package used are org.apache.jena.rdf.* is used. Model class is used to read the Ontology from file. Query class is used to create the Sparql Query. ExecSelect function is used to execute the query and fetch results.

```java

String query1="PREFIX+r

""SELECT distinct ?x ?p ?o WHERE {(?x rdf:subClassOf* my:"+ontologyConcept+
" . ?x ?p ?o) UNION (?x rdfs:subClassOf my:"+ontologyConcept+
" . ?x rdf:type ?class . ?x ?p ?o)" + "UNION ("+"?x owl:unionOf my:"+ontologyConcept+
+ "UNION ("+"?x owl:intersectionOf my:"+ontologyConcept+
+ "UNION ("+"?x owl:complementOf my:"+ontologyConcept+
+ "UNION ("+"?x owl:disjointWith my:"+ontologyConcept+
+ "}) order by ?x;"
```

Figure 5.9 Jena API and Sparql Usage

3) Statistical Techniques used with and without Ontology:

a) Latent Semantic Analysis (LSA): The modified LSA algorithm is implemented with and without Ontology. The use with Ontology is shown in Figure 5.10. In the figure, all the words in Ontology that belong to the concept are used to create submodel answers. Then LSA is used to find presence of each submodel in student answer by changing the value of q vector. This is done by adding only those terms weight in q that is present in the submodel concept.
Figure 5.10 Code to implement LSA with Ontology

```java
String smallModelAnswers1[] = setMasterTermVector(smallModel1,"model",null); // create small model from Ontology concept for each concept
String []masterTermVector1=setMasterTermVector(Answer,"concept",smallModelAnswers1); // count the term vector
double answerTermFrequency1[]=[];
System.out.println("frequency calculation");
answerTermFrequency1 = setAnswerTermFrequency(masterTermVector1,line1);
System.out.println("Calling LSA");
//System.out.println(model);
double dd1=0.0;
int index1=0;
double [][]cor_LSA1=new double[keys1.size()][line1.length]; // keys is a multimap holding the ontology map (multimap class is available in guava
//corLSA1 variable is used for holding correlation of each answer with each concept.
int answerNo = 0;
// for all concepts run the following loop
for(String k1:keys1)
{
    //System.out.println(k1 + " = "+multiMap.get(k1));
    String smallModelQuery1[]= multiMap.get(k1).toString().split("_
    if(index1==0)
        cor_LSA1[index1] =calculateSVD(answerTermFrequency1, masterTermVector1, proxy, processor,smallModelQuery1,1);
    else
        cor_LSA1[index1] =calculateSVD(answerTermFrequency1, masterTermVector1, proxy, processor,smallModelQuery1,0);
    //category1[index1]=k1;
    index1++;
}
```

b) Bilingual Evaluation Understudy (BLEU): The modified BLEU algorithm is tested both with and without Ontology using a similar scheme as shown for LSA.
c) Generalized LSA (GLSA): Generalized Latent Semantic analysis is implemented in Java Programming Language. The difference between GLSA and LSA is that of n-grams. None of the pre-processing steps is performed in GLSA. The word phrases are of length n are formed using sliding window as shown in Figure 5.11. The ngram size is varied between 2 to 4 in the first code snippet. This value of ngram is passed to callGLSA function. In callGLSA function, ith student’s answer is tokenized. Then sliding window is implemented to generate ngrams of specified size. The LSA is performed with phrases by document matrix. Theoretically the GLSA method is improvement over LSA as it looks for phrases and maintains the word order. However, the presence of exact phrases in student answer is difficult to find. It does not consider the synonyms of words. The size of matrix generated becomes even larger than term by document matrix. The GLSA algorithm is implemented with and without ontology.

```
System.out.println("Calling GLSA with ontology");
GLSA g1=new GLSA();

double[][] value_SYN_LSA1=new double[4][Answer.length()];
for(int ngram=2;ngram<4;ngram++)
{
    System.out.println("ngram = "+ngram);
    value_SYN_LSA1[ngram-2]=g1.callGLSA(model2, (float)marks, Answer, myAgent.ngram);
}

-------------
```

```
String[]terms = answers[i].split(" ");
answers[i]=" ";
int end=2;
for (int start=0;start<terms.length ; start++)
{
    end=start;
    String sub=terms[start];
    for(int j=1;j<ngram;j++)
    {
        while(end < start+j)
        {
            end=end+1;
        }
        if(end < terms.length)
        {
            sub=" "+terms[end];
        }
        answers[i]=answers[i]+" "+sub;
    }
}
```

Figure 5.11 Code Snippet for Ngram generation for GLSA

d) Maximum Entropy (MaxEnt): The Maximum Entropy package is downloaded from sourceforge website. MaxEnt is used both with and without Ontology. Without Ontology the model answers are given to maxent and based on these model answers it classifies the new student answers as shown in Appendix D. With Ontology, instead of
classifying the complete answer, individual sentences of student answer are classified and matched to ontology concepts. The scheme is similar as that in LSA. The code snippet showing use of Maximum Entropy with Ontology is shown in Figure 5.12 (a-b). In Figure 5.12 (a) the program snippet for creating category file is shown. First, the concepts are extracted from Ontology. Second, these concepts are written to a category file. In Figure 5.12 (b) this category file is used to create model for evaluation and students answer sentences are categorized into relevant category classes.

```java
// fetch words from on ontology
SubjectiveEvaluation_OntoEvaluation_sparql = new SubjectiveEvaluation_OntoEvaluation_sparql();
String []ontowords = s.fetchResultFromOntology(prefix, fileWithPath, concept);
if(ontowords == null){
    System.out.println("no words could be fetched from ontology");
    System.out.println(prefix + "+" + fileWithPath + "=" + concept);
    return;
}
// count words in ontology are not blanks
String model2=""; System.out.println("this is after fetch"); int count=0;
for(String a:ontowords){
    if((a.length() > 0) && (a.length() <= 1)) {model2 = model2 + a; count+; }
}
if(count <= 4) {System.out.println("not enough words in ontology ... so quitting from ontoMax");return;}
String [] modelList = model2.split("\|\|\|\|\|\|\|\|");
// write ontology to category file
String examiner = System.getenv("EXAMINER_HOME"); if(examiner.contains("\"\")) examiner=""; examiner="";
String CategoryFileContents=""; FileWriter fw;
try {
    fw = new FileWriter(examiner + "\maxOnetoFile.txt");
    Multimap<String, String> multiMap = ArrayListMultimap.create();
    for(String x: modelList){
        String keyval = x.split("\"\"");
        if((keyval[0] != null) && (keyval[0].length == 2)) {
            String ss = keyval[0] + "+" + keyval[0].replace(\_\_\_\_, "+keyval[1]; ss = ss.concat(\"\r\n\"");
            CategoryFileContents = CategoryFileContents + ss; multiMap.put(keyval[0], keyval[1]);
        }
    }
    CategoryFileContents = CategoryFileContents.substring(0, CategoryFileContents.lastIndexOf("\r\n"));
    fw.write(CategoryFileContents);
    fw.close();
}
```

Figure 5.12 (a) Code snippet to create category file with Ontology

### 5.3.4. Objective Evaluation

The implementation of Objective evaluation is shown in Figure 5.13. Multiple-choice-questions are evaluated using matching, Fill-in-the blanks are evaluated by finding synonyms using WordNet and then matching the student words with keyword and its synonyms list.

The One-liner answers are evaluated using BLEU technique with Ontology developed for subjective evaluation.
private double[] callMCQ(String model, double marks, String[] answer)
{
    double res[] = new double[answer.length];
    for(int a=0;a<answer.length;a++)
    {
        if(answer[a].equals(model)) res[a]=marks;
        else res[a]=0;
    }
    return res;
}

private double[] callBlank(String model, double marks, String[] answer)
{
    double res[] = new double[answer.length];
    for(int a=0;a<answer.length;a++)
    {
        if(answer[a].equals(model)) res[a]=marks;
        else{
            String arr = fetchDictionarySynonym(model);
            for(String s:arr.split(" "))
            {
                if(answer[a].equals(s)) res[a]=marks;
                else res[a]=0;
            }
        }
    }
    return res;
}
5.4. Implementation details of Practical Evaluation Module

5.4.1. Tools used

The implementation of Practical Evaluation module is performed by writing code in Java Language using several Tools. There are four steps in evaluation: compilation, testing, metrics and semantic similarity. External tools and libraries are also used wherever available. The tools are discussed in more detail in Appendix E. The list of tools used in Practical module is:

- Turbo C++ and DosBox: These are used for compiling student submitted C and C++ programs. They generate syntax error if any in student program. DosBox is used as Turbo C++ does not work on Windows 8 platform.
- JDK: It is used to compile and execute java programs of students.
- CUnit: It helps in testing the C programs with pre-specified data. The test cases are written by examiner in CUnit API.
- CppUnit and MingW: It is used for testing C++ programs against pre-specified data. MingW is used as CppUnit uses GCC compilers for C++. The test cases of CppUnit are written in CppUnit API.
- JUnit: For testing of java programs by writing test cases in JUnit API.
- C and C++ Code Counter (CCCC) tool: Used to calculate metrics of C, C++ and Java.
- Frama-C: It is tool for generating Program Dependence Graphs (PDG) for C Language.
- SDG API Library: Used to generate PDG for Java Programs.

5.4.2. Implementation of Evaluation Steps

The steps in Practical evaluation are implemented as below:

1) Compilation: The standard compilers of programming language are used for finding the syntax correctness. The Used Compilers are Turbo C++ 3.0 and Java Development Kit 1.7. The Turbo C++ compiler does not work on windows so DosBox 0.74 Terminal Emulator is used, which gives DOS like environment to Turbo C++. The code snippet showing invocation of compilers from Java Program is shown in Figure 5.14. The ‘command’ variable is used to set the command to be executed. Java ‘Runtime’ class is used to invoke the compiler. It was first called directly under main process. Later thread
class was used so that endless loop could be detected. ‘Nanotime’ function is used to keep track of time taken in executing. If time is more, program is aborted. Dosbox is used for running C programs. The ‘exec.sh’ shell script used for executing C++ programs in MingW. The exec.sh is given in Appendix E.

```java
if(language.equals("c")) {
    exefile=filenameCompile[index].replace(".c",".exe");
    command="dosbox -c \""cd tc\bin\" -c \""""
        +exefile+index\" + cutest.c cutest-1.c ""
        +filenameCompile[index]+" > fileCO+index+".txt \" -c \""file\"+index++".exe > outTO"+index+
        ".txt\" -c \"exit\"">overall.txt";
}
else {
    command= MINGW_HOME+"\msys\1.0\bin\sh.exe "+CPPUNIT_HOME+
    +"\examples\simple\exec.sh "+
    +""+filenameCQ[index]+" "%filenameEQ[index];
    FileWriter f= new FileWriter(filenameM[index]);
    f.write(Answer[index]+".replace("","",""a")");
    f.close();
    FileWriter f2= new FileWriter(CPPUNIT_HOME+"\examples\simple\ExampleTestCae.cpp");
    f2.write("\include\""+filenameCompile[index]+"\"a"");
    f2.write(TestMethod);
    f2.close();
}
//Process child = Runtime.getRuntime().exec(command,null);
long start = System.nanoTime();
MyThread th = new MyThread(command);
th.start();
try {
    Thread.sleep(5000);
    if(th.isAlive()) {
        System.out.println("Program taking long to execute");Thread.sleep(5000);
        if(th.isAlive()) {
            th.term();th.stop();
            System.out.println("endlessloop detected");
            reportedErrors[index]=0;actualErrors[index]=0;
            TestPassed[index]="endless"
            long end = System.nanoTime();
            time[index] = (end - start)/1000000
            continue;
        }
    } // if thread is still alive
}
```

Figure 5.14 Code Snippet showing command to call compilers from Java Program

After the compilation, if there are no errors program moves to next step of evaluation. If there are errors, then code given in Figure 5.15 is used to find redundant errors. Whenever syntax mistake is present in any given line, compiler can shows errors in neighboring lines also, if same dependent variables and functions are used. The errors for adjacent line numbers curbed using code written in Java Programming Language.
This code is used for Java as well as C programs, for C++ GNU GCC compiler used in MingW generates only one error at a time so this step is not required.

```java
// code to find redundant errors
actualErrors[index]=0;
if(reportedErrors[index]>0)
{
    // variable Output holds the compiler error output report separated on newline.
    // file1.c is the student program file.
    String key="Error file1.c ";
    int flag=0;
    aa: for(int i=2;(i<output.length);i++) // loop for all error lines in output
    {
        // if error is not in student file continue to next line
        if(!output[i].contains(key)) continue;
        actualErrors[index]++;// Count the error in actual errors.
        // split the error line to extract line number of error shown
        String arr[]=output[i].substring(key.length(),output[i].length()).split(";");
        int ilineno=Integer.parseInt(arr[0].trim());
        int lineno=ilineno;i;
        // Check for all following errors if line number is same or one greater than
        // ignore the error else continue in loop
        for(j=i+1;((j<output.length)&&(output[j].contains(key)));j++)
        {
            String arr1[]=output[j-1].substring(key.length(),output[j-1].length()).split(";");
            String arr2[]=output[j].substring(key.length(),output[j].length()).split(";");
            int jlineno=Integer.parseInt(arr2[0].trim());
            if(arr1[1].contains(arr2[1]))
                lineno=jlineno;
            else if((lineno==jlineno)||(lineno+1==jlineno)||(jlineno+1==lineno))
                if((j>i+2)&&(lineno!=jlineno))
                    if(output[j].contains(key))
                        i=j-1;continue aa; }
            else
                i=j-1;continue aa; }
        }
        i=j-1; // if none of the conditions meet then check next line for redundant errors
    }
    // end for
    // if reported errors are zero that means test cases have been executed. Code to find result of test case
```

Figure 5.15 Code Snippet for Compiler Output
2) Testing: After the programs are successfully compiled, testing output is checked. There are unit testing libraries available. They are open source and easy to integrate. The test question writer has to provide the test cases coded using xUnit package. The details of writing XUnit package are given in Appendix E.

The student programs during testing can enter endless loop. This is handled by executing the program in separate thread and if the thread lives beyond a specified time then it is terminated as shown in Figure 5.14.

3) Metrics Calculation: The Metrics calculated are: time complexity, space complexity, lines of code, lines of comment, number of modules and McCabe’s Cyclomatic complexity.

a) Time complexity is measured by keeping the starting time and ending time of student program and then subtracting the two.

b) Space complexity is calculated by counting variables in code using regular expressions and Pattern matcher class of Java Programming language as shown in Figure 5.16.

c) The remaining four metrics are calculated using a tool called C and C++ Code Compiler (CCCC). The CCCC tool generates output in both HTML and XML format. The numeric values are extracted from the XML file using Java Programming Language code shown in Figure 5.17. The packages used are: ‘org.w3c.dom.Document’, ‘org.w3c.dom.*’, ‘javax.xml.parsers.DocumentBuilderFactory’ and ‘javax.xml.parsers.DocumentBuilder’. The document builder class reads the XML file

```java
Pattern pattern = Pattern.compile("\[a-zA-Z][a-zA-Z0-9]*[;]=]+"');
Matcher matcher = pattern.matcher(memCont);
while(matcher.find())
{
    System.out.println(matcher.group());
    if(matcher.group().equals("int")||matcher.group().equals("void")||
        matcher.group().equals("float")||matcher.group().equals("char")||
        matcher.group().equals("long")||matcher.group().equals("short")||
        matcher.group().equals("double"))
        continue;
    allMatches.add(matcher.group().replaceAll("[=,]",""));
}
Multiset<String> multiset = HashMultiset.create(allMatches);
memory[index] = multiset.size();
```

Figure 5.16 Code snippet showing use of Pattern matching for Counting Variables
components as nodes. These are traversed looking for metric values using fixed key names as attribute of the node.

```java
DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
Document doc;
doc = docBuilder.parse(new File("output.xml"));
doc.getDocumentElement().normalize(); // normalize text representation
NodeList nodeList = doc.getDocumentElement().getChildNodes();
int totalPersons = nodeList.getLength();
for(int s=0; s<=nodeList.getLength() ; s++)
{
    if(nodeList.item(s)="/project_summary")
    {
        Element main1 = (Element)firstProp;
        NodeList childNodes = main1.getChildNodes();
        for (int j = 0; j < childNodes.getLength(); j++)
        {
            Node cNode = childNodes.item(j);
            if (cNode.getNodeName().equals("number_of_modules"))
            {
                try
                {
                    nom[j] = Float.valueOf(cNode.getAttributes().getNamedItem("value").getNodeValue());
                }
                catch Numero
                {
                    // Handle the exception
                }
            }
        }
    }
}
```

Figure 5.17 Code snippet to extract Metrics from XML file generated by CCCC tool

4) Program Dependence Graph Generation: The program dependence graphs of student programs are generated using Frama-C tool and SDG API. A sample output of PDG generated by Frama-C for the program in Figure 5.18 (a) is shown in Figure 5.18 (b). In Figure 5.18 (b), the actual links are shown with ‘a’ label, the control links with ‘c’ label and data flow links with ‘d’ label. Code is developed in Java Programming Language to interpret these graphs as shown in Figure 5.19. The alinks, clinks and dlinks are counted, space is allocated and then edges are saved in the allocated space as arrays.

```java
// Input program for Frama-C.
int main()
{
    int i = 15;
    printf("Factorial of %d is %d\n", i, factorial());
    return 0;
}
```

Figure 5.18 (a) Main function in a C program as input for Frama-C
Figure 5.18 (b) Sample PDG of the C Program in Figure 5.16 (a)
5) Graph Comparison:

a) For C Programs: The comparison between the student program graphs and model program graphs is performed by writing code in Java for implementing Rascal algorithm as shown in Figure 5.20. First, the PDG is generated by tool. This graph is parsed and interpreted. Second, the degree of all types of edges namely, control, data and actual flow are calculated using Java code. Last, the Rascal algorithm is implemented. In Figure 5.20, the calculation of $v_{g2}$, $e_{g2}$, $v_{gg}$ and $e_{gg}$ is shown (refer Figure 4.15). The arrays of degree of each vertex are first sorted and then used.

b) For Java Programs: The SDG library is used to generate SDG graph for Java Programs. Difference between SDG and PDG is that SDG is for whole program consisting of PDG for each function. The code snippet for Java Programs is shown in Figure 5.21. The System dependence graph of student and model program is generated and compared with individual functions in ‘Filename’ containing student and model program. In function ‘check’ the SDG graph is searched for functions in student program and Breadth First Traversal of student program is done and nodes in student function are matched with model functions.
int vg2=0, eg2=0;
// calculation of vg2 and eg2 for student answer.
    for(int i=0;i<p.cstu[0].length;i++)
    {
        if(p.cstu[1][i]>0)    { vg2++; eg2=eg2+p.cstu[1][i];}
    }
for(int i=0;i<p.astu[0].length;i++)
    { if(p.astu[1][i]>0)    { vg2++; eg2=eg2+p.astu[1][i];}}
for(int i=0;i<p.dstu[0].length;i++)
    { if(p.dstu[1][i]>0)    { vg2++; eg2=eg2+p.dstu[1][i];}}
// first tier calculation
// In first tier function the vertices of student graph are matched with model program.
int cV[][]=p.firsttierMatrix(p.cmodel, p.cstu); int aV[][]=p.firsttierMatrix(p.amodel, p.astu);
int dV[][]=p.firsttierMatrix(p.dmodel, p.dstu);
int vgg=0, egg=0;
for(int i=0;i<cV[0].length;i++)
    { if(cV[1][i]>0)      { vgg++; egg=egg+cV[1][i];}}
for(int i=0;i<aV[0].length;i++)
    { if(aV[1][i]>0)       { vgg++; egg=egg+aV[1][i];}}
for(int i=0;i<dV[0].length;i++)
    { if(dV[1][i]>0)   { vgg++; egg=egg+dV[1][i];}}
// calculate the sim (similarity score)
double sim= (vgg+((float)egg/2.0))*(vgg+((float)egg/2.0)) /((vg1+((float)eg1/2.0)) *((vg2+((float)eg2/2.0)));
status[index]=sim;

Figure 5.20 Code Snippet for Graph comparison

String answer="no sim";
int count[]=new int[2][];
try {
    Filename=Filename.replace(".java",".class");
    System.out.println(path+"model"++(path+"model\""+Filename));
    // function to generate system Dependence Graph for Java Program
    SystemDependenceGraph lvSystemDependenceGraph = new SystemDependenceGraph(path+"model",
    (path+"model\""+Filename));
    count[0]=check(lvSystemDependenceGraph,path+"model\""+Filename));
    SystemDependenceGraph2 = new SystemDependenceGraph(path,
    (path+"model\""+Filename));
    count[1]=check(lvSystemDependenceGraph2, (path+"model\""+Filename));
} catch(Exception e) {
    System.out.println(e.getMessage()+" ");
e.printStackTrace();
}
if((count[0])!=null)&&(count[1]!=null)&&(((count[0][0])!=0)&&(count[0][1]!=0)&&(count[1][0]!=0)&&(count[1][1]!=0))
    answer= String.format("%s",(((float)count[0][1]/count[0][0])*((float)count[1][1]/count[1][0])*100);
5.5. Multi-agent System Development

The Multi-agent System in MASMEE is developed using Java Agent Development Framework (JADE 4.3). The code for defining agents and exchanging messages between them is shown through an example (Chat Program) in Appendix A.

The two agents namely, Examiner Agent (EA) and Student Agent (SA) are implemented. The functionality and design of each agent is discussed in Chapter 3. The reason for choosing two agents is Examiner Agent is kept free from any student requests and available to examiner for performing tasks like paper setting and evaluation. The Student Agent is used to service student requests. Gateway Agent is used to expose services of Student Agent over the web and interacts with Java Server Pages. It is just a link between students and Student Agent.

The code snippet showing the implementation of Agents is shown in Figure 5.22. In this figure, the Examiner agent services are registered with DFServices and Login Screen is shown. On successful login, the Welcome Screen is shown. All the screens and working of MASMEE system is discussed in Appendix G.

The interactions between the agents are shown in Figure 5.23. The Gateway agent sends messages to student agent for student login, fetch test, submit test and logout requests. The Student agent notifies the Examiner Agent (EA) to perform evaluation. The evaluation techniques are discussed in Chapter 4 and implemented in previous sections of this chapter.

The Agent behavior of Examiner agent is given in Figure 5.24. The behavior in Figure 5.24 is cyclic behavior. It handles the student request to evaluate the student answers. The message received in ‘mt’ is copied to ‘reply’ variable of type ACLMessage. Then the type of message is identified. It must be an ACLMessage.INFORM type with specific conversation ID. The content of message is ‘paperid’ of the test paper whose student responses are to be evaluated. The questions and answers are fetched from the database and the corresponding functions to perform evaluation are called.

The snapshot of Student Agent behavior sending messages to Gateway Agent are shown in Figure 5.25. The student Agent adds a cyclic behavior to communicate with Gateway agent. The user defined parameters passed in message are shown in the figure.
System.out.println("Examiner started");
try {
    db=new OpenConnection();
} catch(Exception e) {
    System.out.println("opening db connection");
    e.printStackTrace();
}
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("examiner");
sd.setName(this.getName());
dfd.addServices(sd);
try { DFService.register(this, dfd); }
catch (FIPAException fe) { fe.printStackTrace(); }

// Create and show the GUI
Login lg= new Login(this);
lg.setVisible(true);
try {
    this.doWait(); if(lg.allowed == 3) this.doDelete();
    if(lg.allowed == 1){
        myWelcomeGui = new WelcomeGui(this);
        myWelcomeGui.setVisible(true);
        myBehaviour = new StudentRequestPerformer();
        addBehaviour(myBehaviour);
    }
}

Figure 5.22 Code Snippet for Implementing Agents

Figure 5.23 MASMEE Agent Communication
Figure 5.24 Examiner Agent Behavior to perform Evaluation
addBehaviour(new CyclicBehaviour(this))
{
    public void action()
    {
        MessageTemplate mt = MessageTemplate.MatchConversationId("Gateway");
        ACLMessage msg = receive(mt);
        try
        {
            if (msg != null)
            {
                System.out.println("in student agent");
                AnswerBean message1 = operationFromMap(msg);
                if (message1 != null)
                {
                    ACLMessage resp = new ACLMessage(ACLMessage.INFORM);
                    resp.setContent("Success");
                    resp.addReceiver(msg.getSender());
                    resp.setConversationId("Gateway");
                    resp.addUserDefinedParameter("receiver", message1.getReceiver());
                    resp.addUserDefinedParameter("username", message1.getUserName());
                    resp.addUserDefinedParameter("password", message1.getUserPassword());
                    resp.addUserDefinedParameter("userid", message1.getUserType());
                    resp.addUserDefinedParameter("opr", message1.getOpr());
                    resp.addUserDefinedParameter("message", message1.getMessage());
                    resp.addUserDefinedParameter("question", message1.getQuestions());
                    resp.addUserDefinedParameter("type", message1.getType());
                    resp.addUserDefinedParameter("subject", message1.getSubject());
                    resp.addUserDefinedParameter("time", message1.getTime());
                    resp.addUserDefinedParameter("dateofexam", message1.getDateofexam());
                    resp.addUserDefinedParameter("answer", message1.getAnswer());
                    resp.addUserDefinedParameter("rollno", message1.getRollno());
                    resp.addUserDefinedParameter("class", message1.getClass());
                    resp.addUserDefinedParameter("session", message1.getAcademicSession());
                    resp.addUserDefinedParameter("1stOpr", message1.getListOfOperations());
                    send(resp);
                    System.out.println("student sent message to "+ msg.getSender().getName());
                }
                else
                {
                    ACLMessage resp = new ACLMessage(ACLMessage.REFUSE);
                    resp.setContent("Fail");
                    resp.addReceiver(msg.getSender());
                    resp.setConversationId("Gateway");
                    resp.addUserDefinedParameter("opr", message1.getMessage());
                    resp.addUserDefinedParameter("message", message1.getMessage());
                    resp.addUserDefinedParameter("1stOpr", message1.getListOfOperations());
                    send(resp);
                    System.out.println("student sent message to "+ msg.getSender().getName());
                }
            }
        }
        catch (Exception e)
        {
            System.out.println("Error in Student Agent Cyclic Behaviour in Action "+ msg.getMes);
            ACLMessage resp = new ACLMessage(ACLMessage.REFUSE);
            resp.setContent("Fail");
            resp.addReceiver(msg.getSender());
        }
    }
}

Figure 5.25 Student Agent Behavior to send message to Gateway Agent
5.6. Web Interface

The web interface is developed using Web Technologies like – Java Server Pages, HTML 5, CSS, Servlet and deployed using Apache Tomcat Server. Snapshots of MASMEE Web interface and User interface are shown in Appendix F. The web pages are designed using above technologies for student to interact with MASMEE Agents. The structure of Web application developed is shown in Figure 5.26. There are five JSP pages and Gateway agent.

![Figure 5.26 Web Application Structure for MASMEE](image)

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. HTML5 introduces elements and attributes that reflect typical usage on modern websites for example `<nav>` (website navigation block), `<footer>` (usually referring to bottom of web page or to last lines of HTML code), or `<audio>` and `<video>` instead of `<object>`.

CSS is designed primarily to enable the separation of document content from document presentation, including aspects such as the layout, colors, and fonts. This separation can
improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content, such as semantically insignificant tables that were widely used to format pages before consistent CSS rendering was available in all major browsers. CSS makes it possible to separate presentation instructions from the HTML content in a separate file or style section of the HTML file.

Java Server Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. Released in 1999 by Sun Microsystems, JSP is similar to PHP, but it uses the Java programming language. To deploy and run Java Server Pages, a compatible web server with a Servlet container, such as Apache Tomcat or Jetty, is required. A Java Server Pages compiler is a program that parses JSPs, and transforms them into executable Java Servlets. A program of this type is usually embedded into the application server and run automatically the first time a JSP is accessed, but pages may also be precompiled for better performance, or compiled as a part of the build process to test for errors.

Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.

Apache Tomcat™ is an open source software implementation of the Java Servlet, Java Server Pages, Java Expression Language and Java Web Socket technologies. The Java Servlet, Java Server Pages, Java Expression Language and Java Web Socket specifications are developed under the Java Community Process. Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2.

Code snippet showing Servlet invoking Gateway agent is shown in Figure 5.27. The ‘JadeGateway.init’ function is used to initiate the Gateway Agent. The AnswerBean class is defined in this work to exchange the messages and data between JSP and Student Agent. The Request and Response objects are used to pass data to JSP from Student Agent.
if(!JadeGateway.isGatewayActive())
{
    try {
        JadeGateway.init("webGateway.MyGatewayAgent",prof2);
        JadeGateway.execute("hello");
    } catch(Exception ex) {
        System.out.println("Gateway not working");
        request.setAttribute("nature", "Gateway Not working");
        request.getRequestDispatcher("login.jsp").forward(request, response);
    }
}

b1=new AnswerBean();
b1.setReceiver("stugateway");
b1.setOpr opr;
switch(opr)
{
    case "login":
    {
        b1.setUsername(request.getParameter("username"));
        b1.setPassword(request.getParameter("passwd"));
        b1.setUserType(request.getParameter("usertype"));
        b1.setClass1(request.getParameter("class"));
        b1.setAcademicSession(request.getParameter("session"));
        b1.setLoginTime(request.getParameter("login"));
        b1.setListOfOperations(b1.getOpr());
        JadeGateway.execute(b1);
        System.out.println("in send message action recieve " + b1.getOpr());
    }
}

Figure 5.27 Code Snippet for invoking Gateway Agent from Servlet.